Decidable Inductive Invariants for Verification of Cryptographic Protocols with Unbounded Sessions

Felix M. Stutz

1st August 2019

Master Thesis

University of Saarland

Department of Computer Science

First reviewer	Prof. Derek Dreyer, PhD
Second reviewer	Prof. Philippa Gardner, PhD
Supervisor	Emanuele D'Osualdo, PhD

Abstract

Cryptographic protocols are distributed programs that are designed to establish secure communication using an insecure channel. The need for secure communication ranges from internet banking over medical devices to e-voting. Failure of this kind of systems does have immense financial and societal consequences. Over the last decades, there has been efforts to apply formal methods to the design and analysis of cryptographic protocols. In general, verification of cryptographic protocols is undecidable and hence there are different approaches to remedy this obstacle. On the one hand, bounding the number of sessions renders many verification problems decidable. On the other hand, there is actually need to prove correctness in the presence of an unbounded number of sessions. Most approaches targetting this problem suffer from over-approximations and incompleteness of their algorithms as a consequence of dealing with an undecidable problem. Moreover, these approaches lack of a characterisation of protocols for which a precise and terminating analysis is guaranteed.

This thesis is part of a systematic study to understand the structure of cryptographic protocols with an unbounded number of sessions/nonces. To this end, we develop a theory of decidable inductive invariants for the rich class of depth-bounded protocols. While protocols are modelled in a variant of the π-calculus, we present a generic intruder model whose requirements are given as a set of axioms.

First, we prove that this generalised version of depth-bounded protocols admits a post-effective completion. This includes a sound and complete finite representation of downward-closed sets of configurations, a way to decide inclusion of two such sets of configurations based on their finite representation as well as a symbolic version of a post-operator that computes all successors — given a finite representation of a set of configurations.

Second, we have implemented a proof-of-concept prototype tool which exemplifies our approach using an intruder model for symmetric encryption. Due to the high complexity of the algorithmics, we also present various algorithmic aspects which are employed in the tool to make it scale. This includes a coarse variant of widening to infer invariants of protocols automatically.

The benchmarks on toy protocols seem promising in the sense that our invariants can be inferred and proven correct automatically in a performance range from seconds to a few minutes. In the future, our methodology of over-approximating the reachable state space of a cryptographic protocols could be incorporated into other approaches to prune the state space to be considered for their verification techniques. Benefits could range from speeding up current analyses to rendering the verification of new properties feasible.

Acknowledgements

A number of people made this thesis possible and I am obliged to all of them.

First, I would like to thank my supervisor Emanuele D'Osualdo. He has been excellent in giving me the opportunity to pursue my own ideas while providing constructive feedback and indicating possible new directions when needed. I highly appreciate his extra-ordinary approach to research in theoretical computer science that combines interesting theoretical problems with practical applications. I am proud to be taught by such a great teacher.

I am also grateful to Prof. Philippa Gardner for her advice and unwavering support in all matters. This thesis was conducted during a research visit at Imperial College London which has been made possible by her and Prof. Derek Dreyer who agreed on officially supervising my thesis.

I acknowledge the special funding I received by the "Evangelischen Studienwerk - Villigst" for this research visit which was enabled by Prof. Gerd Smolka. I am thankful for the support of "Studienstiftung des Deutschen Volkes", "International Max Planck Research School - Computer Science" and "Evangelisches Studienwerk - Villigst" for supporting my (master) studies in various ways.

Meiner Familie danke ich für ihre Unterstützung, die es mir ermöglicht hat, meinen eigenen Weg zu finden und mir geholfen hat, so weit zu kommen. Insbesondere bin ich froh, dass meine Schwester und meine Mutter, mein Vater mit meiner Stiefmutter und meine Großeltern mich in London besucht haben. Eure Besuche haben immer wieder für das nötige Maß an Zerstreuung gesorgt, das auch nötig war, um produktiver und effizienter forschen zu können.

Mein tiefster Dank gilt jedoch meiner Freundin Laura Neuheisel. Deine ungebrochene Unterstützung und Geduld haben maßgeblich dazu beigetragen, was ich erreicht habe. Deine Gabe, mich in jeder Situation aufzumuntern, ist unbeschreiblich und zu wissen, dass Du nie den Glauben an mich verlieren wirst, bestärkt mich immer wieder aufs Neue.

Contents

1 Introduction **9**
1.1 Motivation . . . 9
1.2 The Setting . . . 10
1.3 Our Approach and Security Properties . . . 11
1.4 Related Work . . . 11
1.5 Contributions . . . 13
1.6 Comparison with Related Work . . . 15
1.7 Outline . . . 15
1.8 Attribution . . . 16

2 Formal Models **17**
2.1 Intruder Models . . . 17
2.2 A Calculus for Cryptographic Protocols . . . 19
2.3 Depth-Bounded Protocols . . . 24

3 Ideal Completions for Security Protocols **27**
3.1 Downward-closed Invariants and Security Properties . . . 27
3.2 Depth-Bounded Processes are Well-Quasi-Ordered . . . 30
3.3 Limits and Ideal Decompositions . . . 32
3.4 Decidability of Inclusion . . . 34
3.5 Computing Post-Hat . . . 44
3.6 Invariant for Example 2 . . . 47

4 Algorithmic Aspects **49**
4.1 Incorporation Check . . . 49
4.2 Irreducible Knowledge . . . 52
4.3 Pattern Matching . . . 55
4.4 Finding Candidates for Invariants . . . 66
4.5 Encoding for SMT-Solver . . . 70
4.6 Tight Form . . . 75

5 Evaluation **81**
5.1 Benchmarks . . . 81
5.2 Otway-Rees Protocol . . . 82

6 Related Work **85**
6.1 Invariants . . . 85
6.2 Bounded Number of Sessions . . . 86
6.3 Unbounded Number of Sessions . . . 87

7 Conclusion **89**
7.1 Summary . . . 89
7.2 Future Work . . . 90

List of Figures **93**

List of Tables **93**

Bibliography **95**

Appendix A **99**
A.1 Proof that Forest Encoding Preserves $\sqsubseteq_{\mathsf{kn}}$ . . . 99
A.2 Properties of Knowledge . . . 99
A.3 Properties of $\sqsubseteq_{\mathsf{kn}}$ . . . 100
A.4 Solving the Example in the Introduction . . . 100

Chapter 1

Introduction

1.1 Motivation

The goal of this thesis is the development of sound and complete theoretical foundations for formal and automatic verification of security protocols.

All communications on the Internet are based on protocols that determine the way information shall be exchanged between two or more parties. Security protocols are distributed programs that establish secure connections over insecure channels between participants of a protocol by using cryptography. In times of e-voting, internet banking and block-chain, secure communication is crucial and security protocols are employed everywhere. Failure of these systems does not only have tremendous financial but also societal consequences. Unfortunately, it is routine to discover new flaws in the design of deployed protocols. Surprisingly, there is a range of these flaws that do not require the underlying cryptography to break but simply exploit their misuse. For instance, the Needham-Schroeder protocol was found vulnerable in 1995 — 17 years after publication [Low95].

An Easy Example Let us consider an example illustrating the idea of protocols. It also exemplifies a possible attack. Suppose you want to send a parcel to a friend of yours and the content shall be secret, so only you and her should know the parcel's content. Both of you are given locks and the respective keys to these locks but you both do not share a pair of lock and key: you do not have a lock to which she has the key and neither vice versa. There is a public post service that cannot be trusted. Before sending the parcel you may have agreed on the following procedure where you are denoted by B and your friend by A.

- You lock the parcel with one of your locks L_B and send it to A.
- Your friend A receives the parcel and applies her own lock L_A to it so that the parcel is locked twice. The parcel is sent back to you.
- You receive the parcel, which you cannot open, but you unlock your lock L_B and send it back to A.
- Your friend receives a parcel that is only locked with L_A so that she can open it.

At first, this sounds like a reasonable approach but in fact it is not. Suppose the compromised post service also knows the steps of your protocol. Then, the intruder can apply his own lock L_I instead of L_A. The two locks look indistinguishable to you. You unlock your lock L_B in the next step and the intruder is actually the only one to open the parcel as it is only locked by L_I. Note that we have not required any lock to be broken but the way keys and locks have been used is flawed.

In this thesis, we devise algorithms to automatically and systematically prove the absence of such flaws.

1.2 The Setting

There are two major models that are considered in the context of cryptographic protocols and messages in particular: the *computational* and the *symbolic* model [Bla16]. In the computational model, messages are considered to be bitstrings and hence constructors are functions from bitstrings to bitstrings. While the computational model is good to verify the primitives, it is difficult to handle their use in a full protocol. Therefore, it is usually assumed that the primitives have been separately proven correct, with the computational model for instance, and the symbolic model then allows to reason about the use of primitives in an algebraic way. From this perspective, messages can be built using (basic) terms and constructors which can be applied to arbitrary terms. For instance, one can apply a two-ary encryption constructor that encrypts a message M with a key K: $\mathsf{e}(M)_K$. With the use of equivalence relations, e.g. $=_E$, one can model the possibility to obtain the original message M by decrypting with K: $\mathsf{d}(\mathsf{e}(M)_K)_K =_E M$.

In the symbolic model, the intruder's capabilities are formalised with the so-called *Dolev-Yao* intruder model. While a Dolev-Yao intruder cannot break the cryptographic primitives, it can intercept any message which is sent by the participants of a protocol. It can also drop messages and thereby prevent a message to reach its intended destination. So the intruder can overhear any communication and use the obtained knowledge to crypto-analyse, e.g. decrypt a message for which the key is known, and construct messages consisting of the obtained knowledge or freshly forged basic names, the ground terms of messages.

We will model protocols in a variant of the π-calculus. By its nature, this will be Turing-complete and hence a lot of security problems become undecidable. There are three main sources of infinity when considering cryptographic protocols:

- size of messages
- number of sessions/nonces
- arbitrary interference between sessions

A nonce is an unguessable, unique random string of bits that can be used as fresh encryption key or as an ID to stamp a message. It is straightforward to see that an unbounded size of messages easily leads to Turing-completeness and hence we bound the size of messages. This is a common restriction for automated approaches.

Let us consider *secrecy* as a first basic security property we will be able to verify. Secrecy of a message M can be thought of as the fact that M is never leaked (to the intruder). Considering secrecy of M, one can either try to find an attack, i.e. a scenario where M is leaked, or try to verify that no attack exists, i.e. prove that there is no scenario in which M is leaked. Proving that an attack exists is simpler than verifying the absence of attacks since a concrete attack will only exploit a bounded number of sessions. Deciding non-secrecy for a bounded number of sessions is indeed **NP**-complete [RT01, CCZ10]. So for non-secrecy, considering a bounded number of sessions suffices.

In contrast, to prove secrecy restricting to a bounded number of sessions would severely limit the generality of the verification result. It would potentially cease to hold in the realistic scenario of unbounded number of sessions/nonces. Most (internet) security protocols are not designed to terminate — for instance banking applications are supposed to provide service to many clients at the same time and for an unspecified amount of time — hence it is not realistic to restrict verification results to a bounded number of sessions. On the one hand, our goal is a theory that supports unbounded number of sessions. On the other hand, we want to have a complete and sound theory. To this end, we have to restrict the class of supported protocols since the problem is undecidable in general. We will generalise the fragment of depth-bounded protocols that was firstly introduced in [DOT17] for which decidability of secrecy was proven. Depth-boundedness is a semantic condition on the use of nonces. It does not prohibit the use of unboundedly many nonces in general but restricts to render the problems we consider decidable. The details will be presented in Section 2.3. Other known decidable fragments use syntactic conditions on the use of nonces, e.g. the ones presented in [CCD15, Fröl5].

In this project, we devise algorithms that can prove security properties without the requirement of bounding the number of sessions or nonces.

1.3 Our Approach and Security Properties

We design methods to prove properties of a protocol by automatically constructing inductive invariants. Intuitively, an *invariant* is a property that always holds. In the setting of protocols, a property is an invariant if it holds for every possible reachable configuration of the protocol. Hence, an invariant can be considered to be a set of configurations that over-approximates the reachable state space of a protocol in presence of an intruder. An invariant is *inductive* if every configuration that is obtained by taking one step from a configuration in the invariant, is again in the invariant. In other words, an inductive invariant is closed under taking one step (and hence making a finite number of steps).

We will discuss the building blocks for our theory in Section 1.5 and focus on possible applications. Invariants can either directly entail properties like secrecy or be part of a chain of arguments proving more involved properties like *trace* or *equivalence properties*.

A *trace property* is a property of a, possibly dishonest, execution of a protocol. An instance of trace property is the fact that in every possible run of the protocol a server authenticates the identity of an honest principal of the protocol A. An *equivalence property* [CCD13] considers two traces of a protocol and compares them. For instance, the intruder should not be able to distinguish whether two people have voted for the same party in an election by comparing the two traces produced by the two vote casts.

In this thesis, we will focus on invariant properties, keeping in mind that, in general, complex security properties are often proven by establishing a hierarchy of increasingly involved invariants. For instance, the guarantee that one cannot distinguish whether two people have voted for the same party may hinge on secrecy of the keys used to transfer their vote, which is an invariant. Hence, our theory of invariants can provide important intermediate facts that are needed for a proof.

1.4 Related Work

The application of formal methods to cryptographic protocols has been quite successful. A number of tools have been deployed that are dedicated to catch bugs especially in early design phases. We briefly survey three major approaches to verification of cryptographic protocols in the symbolic model.

1.4.1 (Manual) Invariants

Both, [Mcl95] and [Pau98] have pioneered the use of invariants for verification of cryptographic protocols. However, their approaches are not fully automated, in particular the process of inferring the invariants is done manually.

In [Mcl95], protocols and security properties are modelled as standard linear-time temporal logic. Besides the common model checking techniques that apply, they propose to strengthen the formulas of security properties so that they become (inductive) invariants.

In [Pau98], they establish proof systems, mechanised in Isabelle/HOL, to prove security properties. One first models the protocol as sequence of communication events. A sequence of events forms a trace and proving a security property is done by induction on these traces. One starts by proving that the fact holds for the empty trace and it remains to prove the claim for doing one step starting from an arbitrary trace. The statements and proofs have to be designed manually while the correctness of the proofs can be checked by the theorem prover Isabelle. Proving a complex security property comprises proofs of various intermediate facts which are combined to obtain the goal. One can also assume known properties which have been proven with other methods. Building upon this, one can for instance prove that some event A always precedes event B provided that the key K remains secret in any possible trace.

We now go back to the question how to deal with the different sources of infinity. As explained before, the sources of infinity easily lead to Turing-complete computational models. We hence want to distinguish two different kinds of approaches: the ones bounding the number of sessions and the ones not bounding the number of sessions.

1.4.2 Automatic Tools for Bounded Number of Sessions

Bounding the number of sessions renders a lot of security problems decidable. However, this decidability comes at the price of missing possible attacks. There are several dedicated tools for automated protocol analysis (e.g. DEEPSEC [CKR18], AVISPA [ABB$^+$05], SPEC [TNH16], AKISS [CCCK16]) under a given bound on the number of sessions. Most of them rely on different flavours of model checking techniques. For a more detailed overview, we refer to Section 6.2.

1.4.3 Semi-Automatic Tools for Unbounded Number of Sessions

We compare to the approaches of three tools supporting an unbounded number of sessions: Tamarin, ProVerif and Maude-NPA. We summarise the approaches of ProVerif and Tamarin and deliberately omit Maude-NPA since the support of unboundedness is inspired by ideas of ProVerif [Bla16]. Moreover, we briefly hint at the use of type systems for verification of cryptographic protocols.

ProVerif In ProVerif [Bla16], protocols are modelled in a variant of the π-calculus. They are first translated to a set of Horn clauses. In order to handle the infinite state space, they apply approximations during the translation. A security property can then be converted to a derivability query on this set of Horn clauses. If the query is not derivable, the security property holds. If it is, there might be an attack. There might be spurious attacks in the sense that the query was derivable due to approximations. In the latter case, the result amounts to a semi-decision procedure.

Tamarin In Tamarin [MSCB13], protocols are modelled as multiset rewriting systems. Security properties were originally modelled as temporal first-order logic formulas. Based on this representation, trace properties could be proven. Later on, the mechanism was extended to equivalence properties using ideas from ProVerif. Note that there is a translation from protocols that are modelled in a variant of the π-calculus to the encoding using multiset rewriting systems [KK14]. A special constraint solving technique checks whether the security property, given as temporal first-order logic formula, holds. The process might require interaction by the user who can give helper lemmas to guide the proofs.

Type Systems One can also use type systems to verify cryptographic protocols as done in [DKSH11, CCD15, CGLM17]. Usually types are used to annotate and generalise safe usages of cryptographic primitives. Pursuing this idea, verification of cryptographic protocols is reduced to type checking which can be efficiently solved.

1.5 Contributions

Our developments are based on [DOT17]. Their class of depth-bounded protocols admits infinite state space for protocols and the use of an unbounded number of new names in a well-behaved way that renders the verification problems we consider decidable. The decidability result in [DOT17] was obtained by instantiating a backward search, i.e. starting from a violation of the property to prove, one tries to reach the initial configuration by a backwards search. Whilst being reasonable for a decidability result, the algorithm does not scale in practice. Forward search approaches are known to scale better in practice but require a deeper understanding of the structure of computation. We unravel the structure of depth-bounded computation by showing that depth-bounded protocols are a completion-post-effective class of well-structured transition systems [BFM18]. This requires the development of non-trivial theory.

We want to automate the check whether a set of configurations I of a protocol with initial configuration P_0 is an inductive invariant. That is, checking $P_0 \in I$ and whether the inclusion $\mathrm{post}(I) \subseteq I$ holds where $\mathrm{post}(I)$ contains all successors of configurations in I. We need different ingredients to devise algorithms to do this automatically: finite representations of invariants, a symbolic version of post(-) and an algorithm to check inclusion.

Finite Representations Since we deal with an unbounded number of sessions, it is not straightforward to represent a set of configurations of a protocol. We first define a class of expressions called "limits". For example, let L be a limit representing the configurations in I, which is denoted by $[\![L]\!] = I$. We prove that they are sound and complete for representing possibly infinite (downward-closed) sets of configurations of depth-bounded protocols. We will show that this enables the representation of non-trivial security properties — like secrecy for instance.

Symbolic Post Given such a finite representation L for I, we want to check whether I is an inductive invariant. We have to verify that every successor of a configuration in I, $P \in \mathrm{post}(I)$, is contained in the invariant. As the set of configurations can be infinite, there might be an infinite number of successors. We therefore define a symbolic version of post(-), denoted by $\widehat{\mathrm{post}}$(-), that computes a finite number of sets of successor configurations that are represented as limits: $\widehat{\mathrm{post}}(L) = \{L_0, \cdots, L_n\}$ iff $\mathrm{post}([\![L]\!]) = \bigcup_{i=0}^{n} [\![L_i]\!]$.

Inclusion Check Given the finite number of successor sets as limits $L_0, \cdots, L_n$, we check for each limit whether it is included in the invariant: $\forall 1 \leq j \leq n : [\![L_j]\!] \subseteq I = [\![L]\!]$. This check is also non-trivial as both sets represent an infinite number of configurations. We will present a characterisation of inclusion of two limits. This leads directly to a recursive algorithm to check inclusion.

The algorithms solving the last two challenges are the major theoretical contributions.

All in all, the contributions of this thesis are:

1. a theory of sound and complete downward-closed invariants for the class of depth-bounded protocols
2. a (forward) verification procedure to check inductivity
3. a widening algorithm that can (heuristically) infer inductive invariants
4. the development of techniques to make the algorithms practicable as well as a prototype implementation.

All of these contributions have been formally proven sound.

Widening Verifying a (basic) security property of a cryptographic protocol using our method requires two steps. First, we need to find a candidate for an inductive invariant L. Second, we apply our algorithms to check if L represents an inductive invariant that implies the desired security property. To automate the first step, we will present an algorithm to infer invariants which is a form of widening in abstract interpretation.

Invariants as Certificates One can consider the second step of the procedure, i.e. the one when inductivity and the property are checked, as a certification process. Given the finite representation of an invariant, its inductivity as well as whether the security property holds can be independently checked. Hence, limits can act as correctness certificates for protocols.

Depth-Bounded Concurrent Systems We model depth-bounded protocols in a variant of the π-calculus. In contrast to variants of the *pure* π-calculus, i.e. the ones not supporting cryptographic primitives, our model is parametric on an active intruder model. Our techniques can be applied to the simpler case of the pure π-calculus— yielding new and more direct algorithms to verify depth-bounded concurrent systems, compared to e.g. [WZH10].

Prototype Implementation We implemented a proof-of-concept prototype that implements symmetric encryption and there are plans to extend it to more cryptographic primitives that are supported by the generic intruder model, e.g. asymmetric encryption, hashing and signatures. Even though the theory is direct to implement, we will present different techniques that can be applied to lower the computational effort. These range from preprocessing checks to make an invariant less redundant to techniques generating possible pattern matchings and sound but incomplete simplifications of the inclusion check. Overall, the case study shows that the approach seems to be promising to scale up to realistic protocols.

1.6 Comparison with Related Work

We want to briefly explain in which ways our approach is different to the related work in Section 1.4 and hint at possibilities how our methodology could be used in their approaches.

For methods in which invariants have to be designed manually, we can provide means to automatically infer invariants. These can then be used as intermediate facts to prove more complex security properties as explained before.

In contrast to all approaches assuming a bounded number of sessions, our verification results will also be valid in the presence of an unbounded number of sessions. However, many of these tools supply efficient algorithms to reason about cryptographic primitives. In our development, we axiomatise the intruder model and consider the cryptographic primitives as blackbox. Therefore, the techniques used to handle algebraic properties of primitives might be useful for general and efficient implementations of our approach.

The tools that can handle an unbounded number of sessions typically need to use incomplete over-approximations. Hence, the model might over-approximate the possible behaviour of the actual protocol. It is unclear how to generalise the class of protocols for which a (precise) answer can be given. Moreover, they are not guaranteed to terminate with an answer on all protocols. For ProVerif, it was proven that the procedure terminates for the syntactic class of *tagged protocols* that are shown to be incomparable to the fragment of depth-bounded protocols in [DOT17]. Our focus is to start a systematic study by establishing the algorithmics for the class of depth-bounded protocols so that the supported class of protocols can be characterised from the beginning. Proving more complex security properties could become more efficient as the state space to consider for a proof is pruned. As invariants over-approximate the reachable state space of a protocol, this does not weaken the verification result. In Tamarin, this could mediate the obstacle of providing helper lemmas to some extent for instance.

In contrast to our approach, type checking is working on a rather syntactic level and it is not straightforward to characterise the supported class of protocols. We speculate that our developments could be used to define an expressive class of constraints, which can be integrated in type systems.

1.7 Outline

The remainder of this thesis is structured as follows.

In Chapter 2, we introduce basic terminology and the formal models. First, we give and motivate the axiomatisation for the intruder models. Second, we introduce the cryptographic π-calculus, in which we model the intruder's capabilities in the reduction semantics. Moreover, we define the class of depth-bounded protocols.

Chapter 3 is the place where we establish the theory of ideals and instantiate the ideal completions framework. To start with, we present possible security properties that are supported directly and explain that invariants can be used in a larger context. We show that depth-bounded processes form a well quasi order for a notion of configuration embedding we define. This forms the basis to present the solutions to the three ingredients to check inductiveness: finite representations, inclusion check and symbolic post-operator.

From a theoretic point of view, the results in Chapter 3 yield a direct verification algorithm. However, a practical implementation requires the development of further techniques which we present in Chapter 4. We start with a simplified inclusion check that can be used as a first indicator but is incomplete. We explain how knowledge can be handled algorithmically for the case of symmetric encryption and hint at possible generalisations.

This representation of knowledge is used for the pattern matching mechanism for which we present a neat way to generate all possible pattern matches. Then, we present a way to infer invariants automatically. We also discuss some specific related work targetting similar problems at this point. At last, we show how one could generalise structural congruence for limits in order to obtain less redundant representations of configurations.

In Chapter 5, we give further details of the implementation and a benchmark suite. The explanation of one of the examples in depth concludes the evaluation.

Chapter 6 is dedicated to related work. After a brief discussion of tools supporting a bounded number of sessions, we explain the ideas behind ProVerif and Tamarin as they can handle an unbounded number of sessions. We also hint at possible ways to incorporate our approach into these tools.

In Chapter 7, we summarise our results and give possible directions for future work.

1.8 Attribution

This thesis is mostly based on the technical report [DS19] which is the result of a collaboration with Emanuele D'Osualdo. As a guideline, I will explicitly cite [DS19] in case this part was mostly developed before I joined the project. Our work [DS19] is dedicated to establishing the algorithmics for the decidable fragment proposed in [DOT17]. While the calculus for cryptographic protocols has already been presented in [DOT17], we extended the intruder model to a generic axiomatisation and established the theory for a forward search approach. Chapter 4 is not part of [DS19].

Chapter 2

Formal Models

In this thesis, security protocols are formalised by a variant of the Applied π-calculus. We axiomatise the intruder model instead of giving one specific intruder model. Hence, our approach can be used for any intruder model satisfying these axioms. For instance, the deduction system introduced in [DOT17] is such an intruder model and will be used for examples.

2.1 Intruder Models

We treat cryptographic primitives algebraically as common in Dolev-Yao intruder models [DY83]. To this end, let $\mathcal{N}$ be an enumerable set of *names* $a, b, \dots \in \mathcal{N}$ and let the signature Σ be a finite set of *constructors*, i.e. symbols f with arity $\mathrm{ar}(\mathsf{f}) \in \mathbb{N}$. Names are used to model data, nonces, and encryption keys in an abstract way while constructors can be used to model encryption for instance. We summarise these constructed messages and various measures on them.

Definition 1 (Messages over Σ, their Size and Names)**.** The set of messages over Σ is the smallest set $\mathbb{M}^\Sigma$ that contains all names and is closed under constructors in Σ:

$$\begin{aligned} \mathcal{N} &\subseteq \mathbb{M}^\Sigma && \text{and} \\ \mathsf{f}(M_1, \dots, M_n) &\in \mathbb{M}^\Sigma && \text{if } \mathsf{f} \in \Sigma \text{ with } \mathrm{ar}(\mathsf{f}) = n \text{ and } M_1, \dots, M_n \in \mathbb{M}^\Sigma. \end{aligned}$$

The functions size: $\mathbb{M}^\Sigma \to \mathbb{N}$ and names: $\mathbb{M}^\Sigma \to \mathscr{P}(\mathcal{N})$ are defined as follows:

$$\begin{aligned} \mathrm{size}(a) &:= 1 & \mathrm{size}(\mathsf{f}(M_1, \dots, M_n)) &:= 1 + \max\{\mathrm{size}(M_1), \dots, \mathrm{size}(M_n)\} \\ \mathrm{names}(a) &:= \{a\} & \mathrm{names}(\mathsf{f}(M_1, \dots, M_n)) &:= \mathrm{names}(M_1) \cup \ldots \cup \mathrm{names}(M_n) \end{aligned}$$

Given $X \subseteq \mathcal{N}$ and $s \in \mathbb{N}$, we define $\mathbb{M}_s^{\Sigma,X} := \{M \in \mathbb{M}^\Sigma \mid \mathrm{names}(M) \subseteq X, \mathrm{size}(M) \leq s\}$. We will omit Σ or X when irrelevant or clear from the context.

We write $\mathbb{K}^\Sigma$ for the set of finite sets of messages, $\mathbb{K}^\Sigma := \mathscr{P}_f(\mathbb{M}^\Sigma)$. We might also refer to subsets of the latter as knowledge bases and will write Γ, Γ' for $\Gamma \cup \Gamma'$ while Γ, M stands for $\Gamma \cup \{M\}$.

The calculus for protocols, which will be introduced later in this chapter, will make use of a pattern matching mechanism for communication. There, names may be substituted for arbitrary messages which is the reason to introduce this concept for the axiomatisation of intruder models.

Definition 2 (Substitutions and Renamings)**.** A *substitution* is a finite partial function $\theta\colon \mathcal{N} \rightharpoonup \mathbb{M}^\Sigma$. For the substitution with $\theta(x_i) = M_i$ for all $1 \leq i \leq n$, we write $\theta = [M_1/x_1,\ \dots,\ M_n/x_n]$ which is often abbreviated with $[\vec{M}/\vec{x}]$.
We write $M\theta$ for the application of substitution θ to the message M, and lift the notation to sets of messages in the obvious way: $\Gamma\theta := \{M\theta \mid M \in \Gamma\}$. A substitution θ is called a *renaming of* $X \subseteq \mathcal{N}$ if it is defined on X, θ is injective, and $\theta(X) \subseteq \mathcal{N}$.

Equipped with these formalisms and notations, we present the axioms an intruder model has to satisfy to be applicable for the techniques presented in this thesis.

Definition 3 (Intruder Model)**.** A *derivability relation* for a signature Σ, is a relation $\vdash\ \subseteq \mathbb{K}^\Sigma \times \mathbb{M}^\Sigma$. The pair $\mathbb{I} = (\Sigma, \vdash)$ is an *(effective) intruder model* if $\vdash$ is a (decidable) derivability relation for Σ, and for all $M, N \in \mathbb{M}^\Sigma$, $\Gamma, \Gamma' \in \mathbb{K}^\Sigma$, $a \in \mathcal{N}$:

$$M \vdash M \tag{Id}$$
$$\Gamma \subseteq \Gamma' \wedge \Gamma \vdash M \implies \Gamma' \vdash M \tag{Mon}$$
$$\Gamma \vdash M \wedge \Gamma, M \vdash N \implies \Gamma \vdash N \tag{Cut}$$
$$M_1, \dots, M_n \vdash \mathsf{f}(M_1, \dots, M_n) \quad \text{for every } \mathsf{f} \in \Sigma \text{ with } \mathrm{ar}(\mathsf{f}) = n \tag{Constr}$$
$$\Gamma \vdash M \implies \mathrm{names}(M) \subseteq \mathrm{names}(\Gamma) \tag{Locality}$$
$$\Gamma\theta \vdash M\theta \iff \Gamma \vdash M \quad \text{for any } \theta \text{ renaming of } \mathrm{names}(\Gamma) \tag{Alpha}$$
$$\Gamma, a \vdash M \wedge a \notin \mathrm{names}(\Gamma, M) \implies \Gamma \vdash M \tag{Relevancy}$$

The *knowledge ordering* for $\mathbb{I}$ is the relation $\leq_{\mathsf{kn}}\ \subseteq \mathbb{K}^\Sigma \times \mathbb{K}^\Sigma$ such that $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ if and only if $\forall M \in \mathbb{M}^\Sigma\colon \Gamma_1 \vdash M \implies \Gamma_2 \vdash M$. We write $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$ if $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ and $\Gamma_2 \leq_{\mathsf{kn}} \Gamma_1$.

Unless specified otherwise, we fix an arbitrary effective intruder model $\mathbb{I}$ for the remainder of the thesis and may omit superscripts when clear from context.

Motivation for the Intruder Axioms The first three axioms formalise what intuitively should hold for any deduction relation:

- Axiom (Id): everything in the knowledge base is known.
- Axiom (Mon): adding knowledge does not impede derivations that have been possible before.
- Axiom (Cut): if something is derivable, we can add it to the knowledge base.

The (Constr) axiom guarantees that the intruder can construct messages using all constructors. The (Locality) ensures that derivable messages can only contain names that are present in the knowledge base, hence no new names can be created when deriving a message. The (Alpha) provides soundness in presence of α-renaming, i.e. we can rename basic names. With (Relevancy), we restrict the intruder only to introduce finitely many new names in one protocol step.

Proposition 1. Given $\Gamma_1, \Gamma_2 \in \mathbb{K}^\Sigma$, $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ if and only if $\forall M \in \Gamma_1\colon \Gamma_2 \vdash M$. As a consequence, if $\vdash$ is decidable, so is $\leq_{\mathsf{kn}}$.

Proof. The direction from right to left is straightforward as $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ and $\Gamma_1 \vdash M$ for every $M \in \Gamma_1$.
For the implication, consider a proof tree for $\Gamma_1 \vdash N$ for some message N. We need to prove

$$\frac{M \in \Gamma}{\Gamma \vdash M}\,\textsc{Id} \qquad \frac{\Gamma, (M,N), M, N \vdash M'}{\Gamma, (M,N) \vdash M'}\,\mathrm{P}_L \qquad \frac{\Gamma \vdash M \qquad \Gamma \vdash N}{\Gamma \vdash (M,N)}\,\mathrm{P}_R$$

$$\frac{\Gamma, \mathsf{e}(M)_K \vdash K \qquad \Gamma, \mathsf{e}(M)_K, M, K \vdash N}{\Gamma, \mathsf{e}(M)_K \vdash N}\,\mathrm{E}_L \qquad \frac{\Gamma \vdash M \qquad \Gamma \vdash K}{\Gamma \vdash \mathsf{e}(M)_K}\,\mathrm{E}_R$$

Figure 2.1: Deduction rules for the derivability relation of $\mathbb{I}_{\mathsf{sy}}$

that $\Gamma_2 \vdash N$. We know that $\Gamma_2, \Gamma_1 \vdash N$ by Mon. It holds that $\Gamma_2 \vdash M$ for every $M \in \Gamma_1$ by assumption. Hence, we can apply Cut $|\Gamma_1|$ times and obtain $\Gamma_2 \vdash N$. ☐

By the axiomatisation of the intruder models and hence the derivability relations, we consider the latter to be black box mechanisms. These may therefore range from rewriting systems to deduction systems. For our examples, we adapt the notation of sequent calculi from [TGD10] and define a deduction system for symmetric encryption.

Definition 4 (Symmetric Encryption)**.** Let $\Sigma_{\mathsf{sy}} = \{(\cdot,\cdot), \mathsf{e}(\cdot).\}$ be a signature where (M, N) pairs messages M and N, and $\mathsf{e}(M)_N$ represents the message M encrypted with key N. The intruder model for symmetric encryption is the model $\mathbb{I}_{\mathsf{sy}} = (\Sigma_{\mathsf{sy}}, \vdash)$ where $\vdash$ is defined by the deduction rules in Fig. 2.1.

Proposition 2. The model Σ_{sy} is an effective intruder model.

Proof. We have to show that every axiom of Definition 3 is satisfied by Σ_{sy}.

- Axiom (Id) is a direct consequence of Rule Id.
- Given some derivation $\Gamma \vdash M$, it is straightforward to construct a derivation for $\Gamma, \Gamma' \vdash M$, which only uses messages from Γ. This entails Axiom (Mon).
- The admissibility of cut rule proven in [TGD10] implies Axiom (Cut).
- There are solely two constructors and hence Axiom (Constr) is given by Rules P_R and E_R.
- Axiom (Alpha) holds as all rules are invariant under renamings.
- Axiom (Relevancy) is Lemma 4 in [DOT17] and can be proven by induction on the depth of a derivation.

In every rule, premises solely consist of subterms of messages in the consequence, so $\Gamma \vdash M$ is decidable. ☐

2.2 A Calculus for Cryptographic Protocols

When modelling cryptographic primitives, it is common to deal with constructors and destructors for messages. But our running example for the intruder model as given in Definition 4 is restricted to constructors. This originates in the fact that we model "destruction" by pattern matching in the calculus we formalise now.

Definition 5 (Calculus for Cryptographic Protocols)**.** Let $\mathcal{Q}$ be a finite signature of process names. Each process name $\mathsf{Q} \in \mathcal{Q}$ is associated with a fixed arity $\mathrm{ar}(\mathsf{Q}) \in \mathbb{N}$. A protocol specification consists of an initial process P and a finite set Δ of definitions of the form $\mathsf{Q}[x_1, \dots, x_n] := A$, with $\mathrm{ar}(\mathsf{Q}) = n$, where the syntax of P and A follows the grammar:

$$P ::= \mathbf{0} \ | \ \nu x.P \ | \ P \parallel P \ | \ \langle M \rangle \ | \ \mathsf{Q}[\vec{M}] \qquad A ::= \mathbf{in}(\vec{x} : M).P \ | \ A + A \quad \text{(actions)}$$

An action $\mathbf{in}(\vec{x} : M).P$ consists of the *input prefix* $\vec{x} : M$ with its pattern M and the *continuation* P. Pattern matching is formalised using substitutions of bound names to messages and will be used in the reduction semantics.

We use the vector notation $\vec{x} = x_1, \dots, x_n$ for lists of pairwise distinct names and may abuse notation by treating vectors as finite sets. The notation $\nu\vec{x}.P$ is a shorthand for $\nu x_1. \cdots \nu x_n.P$. The names $\vec{x}$ are bound in both $\nu\vec{x}.P$ and $\mathbf{in}(\vec{x} : M).P$. We denote the set of free names of a term P with $\mathrm{fn}(P)$ and the set of bound names with $\mathrm{bn}(P)$. As is standard, α-equivalence can be used to rename bound names to fresh names without changing the structure of a term. Therefore, we require, w.l.o.g., that $\mathrm{fn}(P) \cap \mathrm{bn}(P) = \emptyset$. When nesting restrictions $\nu\vec{x}.\nu\vec{y}.P$, we implicitly assume w.l.o.g. that $\vec{x}$ and $\vec{y}$ are disjoint.

The internal action $\boldsymbol{\tau}$ is an abbreviation for $\mathbf{in}(x : x)$ for a fresh x.

Processes of the form $\mathsf{Q}[\vec{a}]$ are called process calls. We assume that there is at most one definition for each $\mathsf{Q} \in \mathcal{Q}$ and that for each definition $\mathsf{Q}[x_1, \dots, x_n] := A$, it holds that $\mathrm{fn}(A) \subseteq \{x_1, \dots, x_n\}$. Process calls and processes of the form $\langle M \rangle$ are called *sequential.* If $\Gamma = \{M_1, \dots, M_k\}$ is a finite set of messages, then $\langle \Gamma \rangle := \langle M_1 \rangle \parallel \dots \parallel \langle M_k \rangle$.

A subterm Q of a term P is called *active* in P if it is not under a prefix. The process $\langle M \rangle$ is called *active message* and is essentially a degenerate form of the notion of an active substitution in the applied π-calculus [AF01], where we omit the domain of the substitution. Intuitively, an active message is a message output by a process that is known to the environment and therefore also to the intruder.

The set $\mathbb{P}$ consists of all processes over an underlying signature $\mathcal{Q}$. We define $P^0 := \mathbf{0}$ and $P^{n+1} := P \parallel P^n$. The parallel composition $P_0 \parallel \cdots \parallel P_n$ is denoted by $\prod_{i=0}^{n} P_i$.

Let us outline the remainder of this section. First, we introduce structural congruence for processes as is standard in order to talk about runtime configurations of protocol executions and compare them. Second, we define the reduction semantics and hence explain how these configurations can evolve. Third, we adapt structural congruence to obtain knowledge congruence by incorporating the derivability relation of the intruder model.

Structural Congruence We start with the standard α-equivalence, denoted by $\overset{\alpha}{=}$. It is the smallest congruence relation that satisfies $\nu x.P \overset{\alpha}{=} \nu y.P[y/x]$ where $y \notin \mathrm{fn}(P)$. Structural congruence, denoted by $\equiv$, is the smallest congruence relation that respects α-equivalence, is associative and commutative with respect to $\parallel$ and $+$ with $\mathbf{0}$ as the neutral element, and satisfies the standard laws:

$$\nu a.\mathbf{0} \equiv \mathbf{0} \tag{Garbage}$$

$$\nu a.\nu b.P \equiv \nu b.\nu a.P \tag{Exchange}$$

$$P \parallel \nu a.Q \equiv \nu a.(P \parallel Q) \quad (\text{if } a \notin \mathrm{fn}(P)) \tag{Scope Extrusion}$$

All these rules do not modify messages which is the reason why α-equivalence is the only one to change them. The axiom (Alpha) guarantees that α-equivalence preserves derivability. Using $\equiv$, we can get the configurations we are aiming for: every process P is congruent to a process in *standard form*:

$$\nu\vec{x}.\big(\langle M_1 \rangle \parallel \cdots \parallel \langle M_m \rangle \parallel \mathsf{Q}_1[\vec{N}_1] \parallel \cdots \parallel \mathsf{Q}_k[\vec{N}_k]\big) \tag{SF}$$

where every name in $\vec{x}$ occurs free in some subterm, $m, k \in \mathbb{N}$ and $Q_i \in \mathcal{Q}\ \forall 1 \leq i \leq n$. The standard form of P, denoted by $\mathrm{sf}(P)$, is unique up to α-equivalence and commutativity of parallel. Standard forms may be abbreviated by $\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)$ where all the active messages are collected in Γ, and Q is a parallel composition of process calls. We will abuse notation and treat Q as a set of process calls, i.e. $\mathsf{Q}_i[\vec{N}_i] \in Q$.

When talking about messages of a protocol configuration, we refer to $\mathrm{msg}(P)$: Let $\mathrm{sf}(P)$ be the expression (SF), we define $\mathrm{msg}(P) = \{M_1, \ldots, M_m\} \cup \bigcup_{i=1}^{n} \vec{N}_i$. Thus $\mathrm{msg}(P)$ is the set of messages appearing in a term. When $m = 0, k = 0, \vec{x} = \emptyset$, the expression (SF) is $\mathbf{0}$.

These standard forms of shape $\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)$ can be considered to be the runtime configurations. They represent the current state of the protocol at some point in time. We might refer to Γ as knowledge base. All information contained in Γ is known to everyone, also to the intruder. Q consists of single participants $\mathsf{Q}[\vec{N}]$ of the protocol with their private knowledge $\vec{N}$ and local state Q. Let us now explain how these configurations can evolve and how the intruder is modelled.

Reduction Semantics We restrict communication to a single insecure global channel. It is easy to model multiple channels with a pair constructor and fresh names. However, all messages sent on these channels will be public and contribute to the new knowledge base. Firing an input action leads to a new local state of a participant and hence a modified runtime configuration. We have two types of input prefixes: $\mathbf{in}(\vec{x} : M)$ and τ. To fire a non-τ input prefix, the intruder needs to construct a message that matches the pattern M. Note that this enables both legitimate and malicious transactions as the intruder can replay genuine messages. Remember that τ-transitions are modelled by $\mathbf{in}(x : x)$ for a fresh x. This is the reason why they can be fired at any time. As x is fresh, it does not occur in the continuation, leading to the same runtime configuration no matter which name was input.

Prior to defining the reduction relation and the induced traces, we clarify some notation: $\mathsf{Q}[\vec{M}] \triangleq A$ if $\mathsf{Q}[\vec{x}] := A' \in \Delta$ and $A \overset{\alpha}{=} A'[\vec{M}/\vec{x}]$, up to commutativity and associativity of $+$.

Definition 6 (Reduction Semantics)**.** The transition relation $\rightarrow_\Delta$ defines the semantics of our calculus:

$$\frac{\mathsf{Q}[\vec{M}] \triangleq \mathbf{in}(\vec{x} : N).P' + A \qquad \Gamma, \vec{z} \vdash N[\vec{M}'/\vec{x}] \qquad \vec{z} \text{ fresh}}{\nu\vec{a}.(\langle\Gamma\rangle \parallel \mathsf{Q}[\vec{M}] \parallel C) \rightarrow_\Delta \nu\vec{a}.\nu\vec{z}.(\langle\Gamma\rangle \parallel \langle\vec{z}\rangle \parallel P'[\vec{M}'/\vec{x}] \parallel C)} \qquad \frac{P \equiv P' \rightarrow_\Delta Q' \equiv Q}{P \rightarrow_\Delta Q}$$

With the set $\mathrm{traces}_\Delta(P) := \{Q_0 \cdots Q_n \mid P \equiv_{\mathsf{kn}} Q_0 \rightarrow_\Delta \cdots \rightarrow_\Delta Q_n\}$, we collect all the transition sequences from P under Δ. The set $\mathrm{reach}_\Delta(P) := \{Q \mid P \rightarrow_\Delta^* Q\}$ is the set of processes reachable from P. For both, we may omit Δ when it is clear from the context.

Because of (Relevancy), we can assume w.l.o.g. that $\vec{z}$ includes only names that appear in $\vec{M}'$ in the first rule. Traces can be used to trace back how information and local states evolved. With the set of reachable runtime configurations, we will be able to check that objectionable behaviour will not occur.

We alluded to before that "destruction" is modelled by the pattern matching mechanism. Considering the intruder model for symmetric encryption from Definition 4, this means that decryption is done implicitly when firing an action. One problem is that general pattern matching would enable a participant to obtain the key from an encrypted message: the pattern $\mathbf{in}(x, k : \mathsf{e}(x)_k)$ even enables the participant to obtain *both* the key k and the plaintext x. This is unreasonable power for the participants which is originated in a modelling problem that can be solved by restricting pattern matching. Intuitively, one should be able to implement all patterns using the cryptographic primitives. This is why we restrict the calculus to use implementable patterns.

Definition 7 (Implementable Patterns)**.** Let $\vec{x} : M$ be a pattern and $Z = \text{names}(M) \setminus \vec{x}$. The pattern is *implementable*, if, for all substitutions $\theta \colon Z \to \mathbb{M}$, we have $M\theta, Z\theta \vdash y$ for all $y \in \vec{x}$.

Note that Z represents all names the process call knows. Moreover, our decidability results do not rely on implementable patterns. Implementability guarantees that the participants of a protocol do not have more power than the intruder. For instance, $\mathbf{in}(x : \mathsf{e}(x)_k)$ is not implementable.

Example 1 (Pattern for $\mathbb{I}_{\mathsf{sy}}$)**.** Consider the following process call definition for which we do not specify the continuation:

$$\mathsf{Q}[u, w] := \mathbf{in}(y : \mathsf{e}(\mathsf{e}(y)_z, \mathsf{e}(z)_w)_u). \cdots$$

This pattern is implementable. The process call $\mathsf{Q}[u, w]$ knows u and w. Hence, one can decrypt the message and obtain the pair $(\mathsf{e}(y)_z, \mathsf{e}(z)_w)$. This can be split and one can obtain z by decrypting with w. It remains to decrypt y with z. Formally, the set of known names is $Z = \{u, w\}$. For implementability, we have to prove that $(\mathsf{e}(\mathsf{e}(y)_z, \mathsf{e}(z)_w)_u)\theta, Z\theta \vdash y$ for every substitution θ. As argued before, $\mathsf{e}(\mathsf{e}(y)_z, \mathsf{e}(z)_w)_u, Z \vdash y$ and hence it holds for $\theta = \mathsf{id}$. This suffices as $\Gamma \vdash M$ iff $\Gamma\theta \vdash M\theta$ holds for every substitution θ for the symmetric intruder $\mathbb{I}_{\mathsf{sy}}$.

Definition 8 (Induced Transition System)**.** Given an initial process P and a set of definitions Δ, which could be an initial protocol configuration, the reduction semantics induces a transition system. The nodes representing different (congruence classes of) configurations and a relation that indicates whether one can perform a step from one configuration to another. The transition system of P is denoted by $(P, \to_\Delta)$.

The evolution of a protocol is captured by traces and reachable configurations. For two configurations, the set of process calls can be compared easily with set equality or inclusion. For knowledge bases, consider two processes that are not structurally congruent:

$$\langle (a, b) \rangle \parallel \mathsf{Q}[\vec{a}] \not\equiv \langle (\mathsf{e}(b)_a, a) \parallel \mathsf{Q}[\vec{a}].$$

Structural congruence does not capture that $(a, b)\rangle \sim_{\mathsf{kn}} (\mathsf{e}(b)_a, a)$. We incorporate the intruder model by extending the structural congruence.

Knowledge Congruence Intuitively, *knowledge congruence*, denoted by $P_1 \equiv_{\mathsf{kn}} P_2$, is the smallest congruence that includes structural congruence $\equiv$ and captures the intruder model in the following way: $\langle \Gamma_1 \rangle \equiv_{\mathsf{kn}} \langle \Gamma_2 \rangle$ if $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$. Formally, we characterise knowledge congruence in the following way:

$$P_1 \equiv_{\mathsf{kn}} P_2 \iff \text{sf}(P_1) \overset{\alpha}{=} \nu\vec{x}.(\langle \Gamma_1 \rangle \parallel Q) \wedge \text{sf}(P_2) \overset{\alpha}{=} \nu\vec{x}.(\langle \Gamma_2 \rangle \parallel Q) \wedge \Gamma_1 \sim_{\mathsf{kn}} \Gamma_2.$$

Knowledge congruence partitions the set of all processes into congruence classes. The distinguishing feature of these is that for two processes in the same class, the intruder (and the participants) cannot tell these two processes apart, neither by observing the future behaviour nor by derivable knowledge. Formally, if $P_1 \equiv_{\mathsf{kn}} P_2$ then the transitions systems $(P_1, \to_\Delta)$ and $(P_2, \to_\Delta)$ are isomorphic. Since the transitions are indistinguishable, we incorporate knowledge congruence into the reduction semantics and therefore add a third rule in the style of the second one to the reduction semantics:

$$\frac{P \equiv_{\mathsf{kn}} P' \rightarrow_{\Delta} Q' \equiv_{\mathsf{kn}} Q}{P \rightarrow_{\Delta} Q}$$

With knowledge congruence, we capture if two runtime configuration are the same in terms of possible successors. We define a notion of "sub-configuration" for runtime configurations that captures whether the set of possible successors of one process is subsumed by the one of some other process.

Definition 9 (Knowledge Embedding)**.** The *knowledge embedding* relation $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ holds if $P_1 \equiv \nu\vec{x}.(\langle\Gamma_1\rangle \parallel Q)$, $P_2 \equiv \nu\vec{x}.\nu\vec{y}.(\langle\Gamma_2\rangle \parallel Q \parallel Q')$ and $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$.

The following proposition reflects the intuitive relation between $\sqsubseteq_{\mathsf{kn}}$ and $\equiv_{\mathsf{kn}}$.

Proposition 3. *$P_1 \equiv_{\mathsf{kn}} P_2$ if and only if $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ and $P_2 \sqsubseteq_{\mathsf{kn}} P_1$.*

Proof. The direction from left to right follows immediately from the definition. For the "if" direction, let us make precise what the order entails. Let $\mathrm{sf}(P_i) \overset{\alpha}{=} \nu\vec{x}_i.(\langle\Gamma_i\rangle \parallel Q_i)$ for $i = 1, 2$. From Definition 9, it is straightforward that

$$P_1 \sqsubseteq_{\mathsf{kn}} P_2 \text{ iff } \Gamma_1 \leq_{\mathsf{kn}} \Gamma_2, \vec{x}_2 = \vec{x}_1\vec{y}_2 \text{ and } Q_2 = Q_1 \parallel Q_2'$$

for some $\vec{y}_2$ and some parallel composition of process calls Q_2' (which is unique up to commutativity of $\parallel$). In the same way,

$$P_2 \sqsubseteq_{\mathsf{kn}} P_1 \text{ iff } \Gamma_2 \leq_{\mathsf{kn}} \Gamma_1, \vec{x}_1 = \vec{x}_2\vec{y}_1, \text{ and } Q_1 = Q_2 \parallel Q_1'$$

for some $\vec{y}_1$ and Q_1'. Note that, by α-renaming, we can choose the names in both normal forms in a way such that this correspondence is possible. This results in $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$. Let us consider the equations for names:

$$\vec{x}_2 = \vec{x}_1\vec{y}_2 \text{ and } \vec{x}_1 = \vec{x}_2\vec{y}_1 \implies \vec{x}_1 = \vec{x}_1\vec{y}_2\vec{y}_1$$

and hence $|\vec{y_1}| = |\vec{y_2}| = 0$. Similarly, $|Q_1'| = |Q_2'| = 0$ and hence $\vec{x}_1 = \vec{x}_2$ and $Q_1 = Q_2$. Overall, this entails $P_1 \sim_{\mathsf{kn}} P_2$. □

Theorem 2.1 (Lemma 11 from [DOT17])**.** *Knowledge embedding is a simulation, i.e. for all P, P' and Q, if $P \rightarrow Q$ and $P \sqsubseteq_{\mathsf{kn}} P'$ then there is a Q' such that $P' \rightarrow Q'$ and $Q \sqsubseteq_{\mathsf{kn}} Q'$.*

Example 2. We present a first toy protocol, which can be used to establish a new session key K between two participants A and B who both know the trusted server S.

$$\begin{array}{lrl}
(1) & A \rightarrow S: & N_A, B \\
(2) & S \rightarrow B: & \mathsf{e}(K)_{(N_A,K_{AS})}, \mathsf{e}(K)_{K_{BS}} \\
(3) & B \rightarrow A: & \mathsf{e}(K)_{(N_A,K_{AS})}, \mathsf{e}(N_B)_K \\
(4) & A \rightarrow B: & \mathsf{e}(N_B)_{(K,K)}
\end{array}$$

The protocol can be encoded as follows:

$$\begin{aligned}
\mathsf{S}[a,b,k_{as},k_{bs}] &:= \mathbf{in}(n_a : (n_a,b)).\nu k.\big(\langle \mathsf{e}(k)_{k_{bs}}\rangle \parallel \big\langle \mathsf{e}(k)_{(n_a,k_{as})}\big\rangle \parallel \mathsf{S}[a,b,k_{as},k_{bs}]\big)\\
\mathsf{A}_1[a,b,k_{as}] &:= \boldsymbol{\tau}.\nu n_a.(\langle (n_a,b)\rangle \parallel \mathsf{A}_2[a,b,k_{as},n_a] \parallel \mathsf{A}_1[a,b,k_{as}])\\
\mathsf{A}_2[a,b,k_{as},n_a] &:= \mathbf{in}\big(k : \mathsf{e}(k)_{(n_a,k_{as})}\big).\mathsf{A}_3[a,b,k_{as},k]\\
\mathsf{A}_3[a,b,k_{as},k] &:= \mathbf{in}(n_b : \mathsf{e}(n_b)_k).\langle \mathsf{e}(n_b)_{(k,k)}\rangle\\
\mathsf{B}_1[a,b,k_{bs}] &:= \mathbf{in}(k : \mathsf{e}(k)_{k_{bs}}).\nu n_b.\big(\langle \mathsf{e}(n_b)_k\rangle \parallel \mathsf{B}_2[a,b,k_{bs},n_b,k] \parallel \mathsf{B}_1[a,b,k_{bs}]\big)\\
\mathsf{B}_2[a,b,k_{bs},n_b,k] &:= \mathbf{in}(\mathsf{e}(n_b)_{(k,k)}).\mathsf{Secret}[k]
\end{aligned}$$

Each principal is modelled by several process calls - each for one step. For instance, $\mathsf{A}_1[\text{-}], \mathsf{A}_2[\text{-}]$ and $\mathsf{A}_3[\text{-}]$ model the three steps of A. We may also omit [-] in this example. Let us assume to start with the following initial configuration:

$$P_0 = \nu a,b,k_{as},k_{bs}.(\mathsf{S}[a,b,k_{as},k_{bs}] \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{B}_1[a,b,k_{bs}] \parallel \langle a,b\rangle).$$

Let us explain an honest run of the protocol. The protocol is initiated by A_1 that sends a new nonce n_a to the server S. Since we assume one insecure channel, we do not annotate the messages with the intended recipient. This does not restrict expressivity in general as we can model channels with a pairing operator. The server receives n_a (or any other message the intruder may generate). It generates a fresh key k and outputs it twice: first, encrypted with k_{bs}, the long-term key shared by B and S; second, encrypted with the pair (n_a, k_{as}). Note that it is possible to use non-atomic encryption keys. In the Alice&Bob-notation, both these messages are sent to B of which it simply forwards the one intended for A. Due to our general insecure message channel, we only model the reception of the message relevant to B with B_1 and can assume that the message intended for A will eventually be fired by the reduction semantics (which also models the intruder). When B receives the new key k, a new nonce n_b is generated which is used to challenge A. Mastering this challenge amounts to sending n_b back but encrypted by (k,k). On reception of $\mathsf{e}(n_b)_{(k,k)}$, B is convinced that k can be used as new session key which is modelled by $\mathsf{Secret}[k]$. This process call ensures that k is not derivable by the intruder in any run of the procotol. To this end, we assume the definition $\mathsf{Secret}[k] := \mathbf{in}(k).\mathsf{Leak}[k]$ in which the continuation $\mathsf{Leak}[k]$ can only occur if k can be derived. Hence, we can ensure that the session key k is safe to use if $\mathsf{Leak}[k]$ does not occur in any reachable process.

2.3 Depth-Bounded Protocols

We generalise the definition of *depth-bounded protocols*, first introduced in [DOT17], by incorporating the more general intruder model. Our goal is a sound and complete theory of invariants for this class of protocols, which we achieve by the means of ideal completions as presented in the next chapter.

Bounding Message Size The motivation for this class of protocols is the possibility to reason about an unbounded number of sessions, new data etc. that is modelled with name restrictions in our calculus. Nevertheless, we are dealing with a Turing-complete model that renders all verification problems undecidable. Let us briefly recall the discussion from [DOT17]. Roughly speaking, the problem can be reduced to the halting problem for two-counter Minsky machines. Two-counter Minksy machines are Turing-complete and consist of two counters which can be incremented, decremented and checked for zero. Intuitively, these two counters can be modelled by two messages of the form $\mathsf{e}(\mathsf{e}(\mathsf{e}(c_0)_{c_1})_{c_2})_{\dots}$ whose length is the current value of the counter. As common in the literature, we restrict the message size by a bound s. Let $\mathbb{S}_s := \{P \in \mathbb{P} \mid \forall M \in \mathrm{msg}(P)\colon \mathrm{size}(M) \leq s\}$ be the set of processes containing messages of size at most s. For some set A, we write A^* for the set of sequences of elements from A. The set of processes reachable from P while respecting a size bound s is the set $\mathrm{reach}^s_\Delta(P) := \{Q \mid P \cdots Q \in \mathrm{traces}_\Delta(P) \cap \mathbb{S}^*_s\}$. Note that this definition does not prohibit the use of bigger messages in the pattern matching but only in the processes of a trace. In order to turn towards a decidable fragment of our calculus, we restrict the size of messages. However, this does not suffice as there are ways to model counters using messages of finite size [DOT17].

Definition 10 (Depth)**.** The *nesting of restrictions* of a term is given by $\mathrm{nest}_\nu\colon \mathbb{P} \to \mathbb{N}$:

$$\begin{aligned}\mathrm{nest}_\nu(\mathsf{Q}[\vec{a}]) := \mathrm{nest}_\nu(\langle M\rangle) := \mathrm{nest}_\nu(\mathbf{0}) &:= 0\\ \mathrm{nest}_\nu(\nu x.P) &:= 1 + \mathrm{nest}_\nu(P)\\ \mathrm{nest}_\nu(P \parallel Q) &:= \max(\mathrm{nest}_\nu(P), \mathrm{nest}_\nu(Q)).\end{aligned}$$

The *depth* of a term is defined as the minimal nesting of restrictions in its knowledge congruence class, $\mathrm{depth}(P) := \min\{\mathrm{nest}_\nu(Q) \mid Q \equiv_{\mathsf{kn}} P\}$.

We illustrate the meaning of Definition 10 and Lemma 1 with an example.

Example 3. $P = \nu a, b, c.(\langle a\rangle \parallel \langle \mathsf{e}(b)_a\rangle \parallel \langle \mathsf{e}(c)_b\rangle \parallel \langle c\rangle)$ which has $\mathrm{nest}_\nu(P) = 3$. The process P is knowledge-congruent to $Q = (\nu a.\langle a\rangle \parallel \nu b.\langle b\rangle \parallel \nu c.\langle c\rangle)$ which has $\mathrm{nest}_\nu(Q) = 1$; this gives us $\mathrm{depth}(P) = \mathrm{nest}_\nu(Q) = 1$. Although $\mathrm{bn}(Q) = \{a, b, c\}$, by α-renaming all names to x we obtain $Q' = (\nu x.\langle x\rangle \parallel \nu x.\langle x\rangle \parallel \nu x.\langle x\rangle)$ which has the property $|\mathrm{bn}(Q')| \leq \mathrm{nest}_\nu(Q) \leq \mathrm{depth}(P)$.

Thinking of the syntax tree of a term, the nesting of restrictions determines the maximal number of name restrictions on any path from the root to a leaf. By α-renaming, we can reuse names in different branches which leads to the following observation.

Lemma 1 ([DOT17])**.** Every Q is α-equivalent to a process Q' such that $|\mathrm{bn}(Q')| \leq \mathrm{nest}_\nu(Q)$.

More generally, Lemma 1 says that for a process of depth k, we can always find a knowledge congruent process that uses at most k unique names, by reusing names in disjoint scopes.

Let $\mathbb{D}^X_{s,k} := \{P \in \mathbb{S}_s \mid \mathrm{fn}(P) \subseteq X, \exists Q \in \mathbb{S}_s\colon Q \equiv_{\mathsf{kn}} P \wedge \mathrm{nest}_\nu(Q) \leq k\}$ be the set of processes of depth at most $k \in \mathbb{N}$, with free names in X, and messages not exceeding size s. The set X solely determines the free names that might be present in processes in $\mathbb{D}^X_{s,k}$. In our use case, we start with a protocol specification, i.e. an initial process P_0 for instance with its free names $\mathrm{fn}(P_0)$. The reduction semantics will never lead to omitting name restrictions at any step. Hence, every reachable process P can only have free names that are already present in P_0: $\mathrm{fn}(P) \subseteq \mathrm{fn}(P_0)$. This is why we omit the superscript X in our context. Intuitively, we call a protocol with its initial configuration P_0 *depth-bounded* if its reachable state space is included in $\mathbb{D}_{s,k}$ for some k.

Definition 11. For some $s, k \in \mathbb{N}$, we say the process P is (s, k)*-bounded* (w.r.t. a finite set Δ of definitions) if $\mathrm{reach}^s_\Delta(P) \subseteq \mathbb{D}_{s,k}$, i.e. from P only processes of depth at most k can be reached, in traces respecting the size bound s.

Note that the two bounds s and k are very different in nature. Bounding the size of messages restricts the traces to be considered: this may result in ignoring some behaviour, i.e. if it is only exhibited by using messages exceeding size s. When considering the rest of the behaviour, we then check if all the reachable processes have depth at most k.

If so, the initial process is (s, k)-bounded. This definitions has two major implications for our analysis. First, when a (s, k)-bounded process is considered, the analysis is only sound with respect to bugs that can be triggered by messages that do not exceed the size bound s. Second, checking whether an initial configuration is (s, k)-bounded cannot be done statically. One the one hand, it suffices to provide a single (message size respecting) trace leading to a process exceeding depth k to show that a configuration is not (s, k)-bounded. On the other hand, proving (s, k)-boundedness requires a characterisation of the set of reachable processes. In Section 3.1, we explain how our theory of invariants can be used to prove boundedness.

Moreover, the depth-bound k and its existence depends on the bound on the message size b.

Lemma 2 (Monotonicity of Existence of k)**.** Let $P \in \mathbb{P}$ be a process and $s_1 \in \mathbb{N}$. Assume P is not (s_1, k)-bounded for any $k \in \mathbb{N}$. Then, P is not (s_2, k)-bounded for every $s_2 \in \mathbb{N}$ s.t. $s_2 \geq s_1$ and any $k \in \mathbb{N}$.

Proof. Let $k \in \mathbb{N}$ be any number. The fact that P is not (s, k)-bounded means that there is a process $P' \in \mathrm{reach}^{s_1}_{\Delta}(P)$ such that $\mathrm{depth}(P') > k$. Obviously, $\mathrm{reach}^{s_1}_{\Delta}(P) \subseteq \mathrm{reach}^{s_2}_{\Delta}(P)$ and hence, this witness can also be found when respecting the bigger size bound s_2. $\square$

Example 4 (Depth-boundedness)**.** In Section 3.6, we show that Example 2 is $(3, 7)$-bounded.

Example 5 (Encryption Oracle)**.** Communication protocols frequently consist of two (or more) parties who want to establish a secure way to communicate and a server, i.e. a party trusted by both (or all) parties to facilitate this way. The latter might comprise a definition of the following shape: $\mathsf{E}[k] := \mathbf{in}(x : x).(\langle \mathsf{e}(x)_k \rangle \parallel \mathsf{E}[k])$. Typically, k is not known to the intruder and x is a nonce. But this automatically leads to unboundedness if the size bound s is chosen in a way so that x can match messages of size greater than 1. This is due to the possibility to construct "encryption chains": the intruder can inject messages (c_i, c_{i+1}) for unboundedly many i, where c_i are intruder-generated nonces. This is basically a type confusion attack and leads to "encryption chains" of the form $\nu k.\nu c_1, \ldots, c_n.(\langle \mathsf{e}(c_1, c_2)_k \rangle \parallel \langle \mathsf{e}(c_2, c_3)_k \rangle \parallel \ldots \langle \mathsf{e}(c_{n-1}, c_n)_k \rangle)$ that are part of the set of reachable configurations. While every single message does not exceed s, the number of name restrictions needed in the same scope can be arbitrarily high (and be chosen by the intruder). Hence, initial protocol configurations are not (s, k)-bounded whenever s allows such encryption chains to evolve.

The pattern presented in this example can be usually modified or constrained to obtain a bounded protocol. The first option exploits the fact that each honest participant of the protocol and the server share a key with the server which can be used to guard the communication and to modify the input action to $\mathbf{in}(x : \mathsf{e}(x)_e)$ for some key e. The second option makes use of a refinement of the message size restriction we will discuss in Section 4.3 in detail. Intuitively, we will be more precise on the size bound assumptions by specifying the bound on the message size for each pattern variable in an input action as introduced in [DOT17]. Moreover, we briefly sketch ideas for a intruder model axiomatisation that can handle the oracle pattern without these patches in Section 7.2.

Chapter 3

Ideal Completions for Security Protocols

This chapter presents the main theoretical contributions of this thesis. Our main theoretical contribution is the fact that the presented (s, k)-bounded protocols are a completion-post-effective class of well-structured transition systems [BFM18]. Let us first outline the procedure induced by this fact and possible applications to security properties. Afterwards, we proceed with instantiating the ideal completions framework in detail and prove the results.

3.1 Downward-closed Invariants and Security Properties

The goal for verification of cryptographic protocols is to prove that a protocol P fulfils some security requirement. These can range from secrecy to trace equivalence. No matter which requirement is to be proven, it is common to establish intermediate results about executions of protocols and combine them to achieve the overall goal. It is easy to see that such an intermediate result might be the fact that some key k is always secret. This kind of property is called invariant and is defined as follows:

Definition 12 (Invariant)**.** An invariant I of P (under definitions Δ and size constraint s) is any set of processes that includes its reachable state space: $\text{reach}^s_\Delta(P) \subseteq I$.

Example 6. Let P be the initial configuration of a protocol so that the free name k is never known to the intruder - in any execution of the protocol. The name k can be considered to be some initial key for instance. Then, $\mathcal{S}_k := \{Q \mid \langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q\}$ which is the set that contains all processes in which k is not public is an invariant for P.

The invariant $\mathcal{S}_k$ is downward-closed by definition, as $\langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q$ and $Q' \sqsubseteq_{\mathsf{kn}} Q$ entails that $\langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q'$. For our analysis, we will focus on the class of $\sqsubseteq_{\mathsf{kn}}$-*downward-closed* invariants.

Definition 13 (Downward-closure and -closedness)**.** Let $X \subseteq \mathbb{P}$ be a set of processes. The $\sqsubseteq_{\mathsf{kn}}$-*downward-closure* of X is defined as: $X{\downarrow} := \{Q \mid \exists P \in X : Q \sqsubseteq_{\mathsf{kn}} P\}$. A set X is said to be $\sqsubseteq_{\mathsf{kn}}$-*downward-closed* if $X = X{\downarrow}$.

The task we have to solve is the following: given an initial protocol configuration P and a candidate for an invariant, i.e. a $\sqsubseteq_{\mathsf{kn}}$-downward-closed set I, prove that $\text{reach}^s_\Delta(P) \subseteq I$. Note that it equivalent to show that $\text{reach}^s_\Delta(P){\downarrow} \subseteq I$. Since P is an initial configuration for some protocol, $\text{reach}^s_\Delta(P)$ consists of processes occurring in traces starting from P. Hence, we know that every possible successor of some $P' \in \text{reach}^s_\Delta(P)$ is also in the latter set. This presumes that the invariants we will be considering have some inductive characteristics.

Definition 14 (post(-) and Inductive Invariants)**.** Let $X \subseteq \mathbb{P}$ be a set of processes. We define the set of processes reachable in one step (respecting the message size bound s) from processes in X to be: $\text{post}^s_\Delta(X) := \{Q' \mid \exists Q \in X, Q \to Q' \in \mathbb{S}_s\}$. The set X is called *inductive* if $X \supseteq \text{post}^s_\Delta(X)$.

We cannot claim that all invariants are naturally inductive but we can restrict ourselves to inductive invariants.

Lemma 3 (Inductive Invariants Sufficient)**.** Let $P \in \mathbb{P}$ be some process and I some $\sqsubseteq_{\mathsf{kn}}$-downward-closed invariant such that $I \supseteq \text{reach}^s_\Delta(P)$. Then, there is an inductive invariant $I' \subseteq I$ such that $I' \supseteq \text{reach}^s_\Delta(P)$.

Proof. The proof is straightforward. We can choose I' to be $\text{reach}^s_\Delta(P)\!\downarrow$ which is downward-closed and inductive by definition.

□

Our strategy to prove that $\text{reach}^s_\Delta(P)\!\downarrow \subseteq I$ for some downward-closed invariant I consists of two steps: Check that $P \in I$ and that I is inductive. We need three ingredients for this idea:

1. a finite representation of downward-closed sets that is recursively enumerable
2. means to decide inclusion of two downward-closed sets based on their representation
3. a symbolic version of post$(-)$, i.e. given some finite representation of D, an algorithm that computes the finite representation of $\text{post}^s_\Delta(D)\!\downarrow$.

In general, downward-closed sets cannot be finitely represented, in particular when considering an unbounded number of sessions. However, we will develop solutions for all the three requirements by restricting the intruder model in a mild way. By establishing these results, we prove that $\mathbb{D}_{s,k}$ admits a *post-effective ideal completion* as presented in [FG09, BFM18].

Separation of State Space The invariant $\mathcal{S}_k$ we introduced in Example 6 is downward-closed. It was basically determined by taking all processes and excluding the behaviour we want to prove absence of: $\mathcal{U}_k := \mathbb{P} \setminus \mathcal{S}_k$. This fact originates in the design and can not be assumed to hold in general. Therefore, we can also characterise $\mathcal{U}_k$ as follows: $\mathcal{U}_k := \{Q \mid \langle k \rangle \sqsubseteq_{\mathsf{kn}} Q\}$. It is well-known that the complement of a downward-closed set is upward-closed which is defined in the obvious way.

Definition 15 (Upward-closure and -closedness)**.** Let $X \subseteq \mathbb{P}$ be a set of processes. The $\sqsubseteq_{\mathsf{kn}}$-*upward-closure* of X is defined as: $X\!\uparrow := \{Q \mid \exists P \in X \colon P \sqsubseteq_{\mathsf{kn}} Q\}$. A set X is said to be $\sqsubseteq_{\mathsf{kn}}$-*upward-closed* if $X = X\!\uparrow$.

Let us sketch the general approach briefly again. Given a protocol specification P, we find an invariant I over-approximating the behaviour of the protocol. Then, we check the absence of undesired behaviour - called U for now - in the invariant: $U \subseteq \mathbb{P} \setminus I$. All invariants we consider are inductive and downward-closed and therefore $\mathbb{P} \setminus I$ is upward-closed. Note that this does not imply that the security properties we can prove are restricted to upward-closed sets but is an observation about how the state space is separated. Now, we turn to possible concrete applications in terms of security properties.

3.1.1 Decidable Properties

Prior to proving Problems 1 to 3, let us explain which kind of security properties can be solved with these means. We restrict ourselves to facts that can be proven directly using the above means in the sense that no other intermediate results are needed. We do not consider properties that are provable when provided directly provable facts as intermediate results. Note that the presented procedures may involve expensive enumeration techniques. This results from the fact that we are establishing decidability rather than computability results. We will explain in Chapter 4 how to adapt these procedures to make them work in practice. Let us start with a non-security property to outline the procedure that can be used as blueprint for the remaining properties.

Deciding (s,k)-boundedness This is rather a property of theoretical interest than a security property. Practically, it comes as a by-product when proving any security property. By finding an invariant, we know that the reachable state space can be over-approximated and is hence (s,k)-boundedness. Note that the process of finding an invariant is highly non-trivial and we will present first attempts to automate this in Section 4.4. From a theoretical point of view, we can come up with a procedure to decide (s,k)-boundedness for a given protocol P that relies on expensive enumeration techniques. The procedure is based on two semi-algorithms that are run in parallel. First, we recursively enumerate all finite representations of invariants and check for each whether (A) it is inductive and (B) it contains P. As soon as we find such a set I, we have proven (s,k)-boundedness of P: $\mathbb{D}_{s,k} \supseteq I \supseteq \text{reach}^{s}_{\Delta}(P)\!\downarrow$. Second, we run an exploration on the set of traces $\text{reach}^{s}_{\Delta}(P) \cap \mathbb{S}^{*}_{s}$. If a process exceeding the depth bound k occurs, we know that P is not (s,k)-bounded. The question arises why these two procedures forming semi-algorithms and their parallel execution suffice. In case $\text{reach}^{s}_{\Delta}(P)$ is (s,k)-bounded, the enumeration of possible invariants will eventually consider $\text{reach}^{s}_{\Delta}(P)\!\downarrow$. The latter satisfies (A) and (B) and the algorithm terminates with a positive answer. In case $\text{reach}^{s}_{\Delta}(P)$ is not (s,k)-bounded, the exploration of traces will eventually consider a process exceeding the depth-bound k and terminate with a negative answer. Combining both cases leads to a complete and sound decision procedure.

Deciding Control-State Reachability and Secrecy Let us start with the property already presented in [DOT17]: secrecy. Roughly speaking, secrecy is the absence of leaks. It can be defined in our calculus using process call definitions: $\mathsf{Secret}[m] := \mathbf{in}(m : m).\mathsf{Leak}[m]$. The latter can only be used by honest participants of the protocol as the property is violated trivially otherwise. More generally, secrecy is an instance of control-state reachability that asks whether the process call $\mathsf{Q}[\cdots]$ is present in any reachable process. To decide control-state reachability, we can apply the presented procedure by augmenting the checks. First, we additionally check in the first semi-algorithm whether $\mathsf{Q}[\cdots]$ is in I. If not, we terminate with a positive answer. If so, we continue. Second, in addition to checking whether a process exceeds the depth-bound, we examine whether $\mathsf{Q}[\cdots]$ is contained. If so, we terminate with a negative answer. More generally, our results imply decidability of (forward) coverability [BFM18], which subsumes secrecy and control-state reachability.

Susceptibility to Known-/Chosen-Plaintext Attacks Consider the encryption oracle from Example 5. Based on its definition, the intruder can send arbitrarily many fresh names to the oracle to encrypt. The observable information is a very large list of pairs of plaintexts and their encryption with the same key k. This results in a higher possibility to break the key k by crypto-analysing them. While this variant is called chosen-plaintext attack, the possibility of only observing plaintexts instead of choosing them is called known-plaintext attacks. Intuitively, it is harder to break the key k with the latter attack. Using invariants that over-approximate the reachable state space of some protocol, we can prove the absence of these attacks. To this end, we simply need to verify that the invariant does not include the downward-closed set $\{ \nu k.(\nu m.(\langle m \rangle \parallel \langle \mathsf{e}(m)_k \rangle))^n \mid n \in \mathbb{N} \}$. The absence of this downward-closed set ensures that the intruder is neither able to observe an arbitrary number of plaintext-ciphertext pairs nor can he even inject the plaintexts by himself. Note that we might obtain invariants containing the latter downward-closed set even though no such attack is possible. On the other hand, if a protocol is *susceptible*, we cannot certify this property which is why our procedure is a semi-decision procedure. Overall, it is remarkable that this property can only be expressed if one considers the case of unbounded number of sessions/nonces.

Remark 1 (Certification and Further Applications)**.** Note that for all three applications, the output will either be a counterexample, which can be used to amend the protocol, or a description of an invariant, which can be independently checked for correctness. Moreover, it can be used to prove further properties - for instance when secrecy for some key has been established. There are also verification techniques exploring the state space of protocols in order to prove security properties. Our invariants can augment these procedures by pruning the search space to be considered.

3.2 Depth-Bounded Processes are Well-Quasi-Ordered

Our goal is to construct finite representations of downward-closed invariants. To this end, we will exploit properties of the algebraic structure of $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$ as done in [DS19].

Definition 16 ((Well) Quasi Order)**.** A relation $\sqsubseteq \subseteq S \times S$ over some set S is a *quasi-order* (qo) if it is reflexive and transitive. An infinite sequence $s_0, s_1, \ldots$ of elements of S is called *good* if there are two indexes $i < j$ such that $s_i \sqsubseteq s_j$. The sequence is called *bad* if it is not good. When the qo $(S, \sqsubseteq)$ has no bad sequences it is called a *well quasi order* (wqo).

We will prove that $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$ is not only a quasi order but a well quasi order by establishing a correspondence between processes in $\mathbb{D}_{s,k}$ and finitely-labelled forests of height at most k. Let us briefly sketch the ingredients for this correspondence. We will define a function $\mathcal{F}[\![\text{-}]\!]_k \colon \mathbb{P} \to \mathbb{F}^Y_{s,k}$ takes a process P with $\mathrm{nest}_\nu(P) < k$ and transforms it into its forest encoding. Conversely, the process that is represented by a single forest in $\mathbb{F}^Y_{s,k}$ will be straightforward. The elements in $\mathbb{F}^Y_{s,k}$ will be represented by nested multisets. Overall, we will exploit the known fact that multisets over a wqo are a wqo which results in the wqo $(\mathbb{F}^Y_{s,k}, \sqsubseteq_{\mathbb{F}})$.

Multisets Let X be a set and $\mu \colon X \to \mathbb{N}$ be a function. The support of μ is defined as $\mathrm{supp}(\mu) := \{x \mid \mu(x) > 0\}$ and we say that μ is finite-support if $\mathrm{supp}(\mu)$ is finite. The set of functions from X to $\mathbb{N}$ with finite support are exactly the multisets of X, $\mu \in \mathcal{M}(X)$. For a quasi order $(X, \sqsubseteq)$; we define the *multiset extension* $(\mathcal{M}(X), \sqsubseteq_{\mathcal{M}})$ – which is also a quasi order – as follows: $\mu_1 \sqsubseteq_{\mathcal{M}} \mu_2$ holds for two multisets $\mu_1, \mu_2 \in \mathcal{M}(X)$ just if there is an injective function $f \colon \mathrm{supp}(\mu_1) \to \mathrm{supp}(\mu_2)$ such that for each $x \in \mathrm{supp}(\mu_1)$, we have $x \sqsubseteq f(x)$ and $\mu_1(x) \leq \mu_2(f(x))$. When X is a finite set ordered by equality, $\sqsubseteq_{\mathcal{M}}$ coincides with multiset inclusion. If $(X, \sqsubseteq)$ is a wqo, then $(\mathcal{M}(X), \sqsubseteq_{\mathcal{M}})$ is a wqo [FG09].

Forests We define a domain of forests $\mathbb{F}^X_{s,k}$ with sequential processes in $\mathcal{B}_s(X)$ as leaves:

$$\mathcal{B}_s(X) := \{\mathsf{Q}[\vec{M}] \mid Q \in \mathcal{Q}, \mathrm{ar}(\mathsf{Q}) = |\vec{M}|, \vec{M} \subseteq \mathbb{M}^X_s\} \cup \mathbb{M}^X_s$$
$$\mathbb{F}^X_{s,0} := \mathcal{M}(\mathcal{B}_s(X)) \qquad \mathbb{F}^X_{s,k+1} := \mathcal{M}\Big(\mathcal{B}_s(X) \uplus \mathbb{F}^{X\cup\{x_{k+1}\}}_{s,k}\Big) \qquad \text{(assuming } x_{k+1} \notin X\text{)}$$

The domain can be considered to be an inductive data structure. The set $\mathcal{B}_s(X)$ contains all sequential processes of $\mathbb{D}^X_{s,k}$. In the base case, forests have height 0 and $\mathbb{F}^X_{s,0}$ are simply a multiset of sequential processes. Forests of height $k+1$ are represented in $\mathbb{F}^X_{s,k+1}$ by a multiset of sequential processes and subforests of height k.

Definition 17 (Definition of Encoding)**.** We use multiset comprehensions $\{\!|\ldots|\!\}$ for this definition. For any $k \geq \mathrm{nest}_{\mathsf{v}}(P)$ we define:

$$\mathcal{F}[\![P]\!]_k := \begin{cases} \emptyset & \text{if } P = \mathbf{0} \\ \{\!|P|\!\} & \text{if } P \text{ is sequential} \\ \{\!|\mathcal{F}[\![Q[x_k/x]]\!]_{k-1}|\!\} & \text{if } P = \mathsf{v}x.Q \\ \mathcal{F}[\![Q_1]\!]_k \oplus \mathcal{F}[\![Q_2]\!]_k & \text{if } P = Q_1 \parallel Q_2 \end{cases}$$

assuming $\{x_1, \ldots, x_k\} \cap (\mathrm{bn}(P) \cup \mathrm{fn}(P)) = \emptyset$. In case this assumption is not met due to bound names, α-renaming is applied implicitly before applying the definition. For $k < \mathrm{nest}_{\mathsf{v}}(P)$, $\mathcal{F}[\![P]\!]_k$ is undefined.

Let us recall an example for a forest encoding from [DS19].

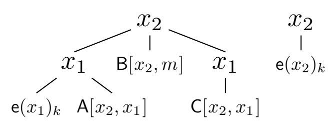
Example 7. We consider the following process:

$$P = \mathsf{v}a.\big(\mathsf{v}c.(\langle \mathsf{e}(c)_k\rangle \parallel \mathsf{A}[a,c]) \parallel \mathsf{B}[a,m] \parallel \mathsf{v}d.(\mathsf{C}[a,d])\big) \parallel \mathsf{v}b.\langle \mathsf{e}(b)_k\rangle$$

We can split P in the following way:

$$\begin{array}{ll} P \;= P_1 \parallel P_2 & \\ P_1 = \mathsf{v}a.(P_3 \parallel \mathsf{B}[a,m] \parallel P_4) & P_2 = \mathsf{v}b.\langle \mathsf{e}(b)_k\rangle \\ P_3 = \mathsf{v}c.(\langle \mathsf{e}(c)_k\rangle \parallel \mathsf{A}[a,c]) & P_4 = \mathsf{v}d.\mathsf{C}[a,d] \end{array}$$

and the forest encoding $\mathcal{F}[\![P]\!]_2 \in \mathbb{F}^{\{m,k\}}_{2,2}$ where restrictions at nesting level k are renamed canonically to x_{2-k} and is represented as follows:



Lemma 4. If $Q \in \mathbb{S}^Y_s$ and $\mathrm{nest}_{\mathsf{v}}(Q) \leq k$, then $\mathcal{F}[\![Q]\!]_k \in \mathbb{F}^Y_{s,k}$.

Proof. By easy induction on the structure of Q. $\square$

In our context, the message size bound s is fixed and given an initial protocol specification P, the set of free names $\mathrm{fn}(P)$ is determined and finite, therefore $\mathcal{B}_s(X)$ has finite cardinality and hence forms a wqo with equality. We define the quasi order $(\mathbb{F}^X_{s,k}, \sqsubseteq_{\mathbb{F}})$ in a natural way as induced by the inductive data structure: for $\mathbb{F}^X_{s,0}$ it coincides with the multiset extension of equality on $\mathcal{B}_s(X)$; for $\mathbb{F}^X_{s,k+1}$ it coincides with the multiset extension of the disjoint union of equality on $\mathcal{B}_s(X)$, and the forest embedding on $\mathbb{F}^{X\cup\{x_{k+1}\}}_{s,k}$. Overall, $(\mathbb{F}^X_{s,k}, \sqsubseteq_{\mathbb{F}})$ is a wqo.

Lemma 5. Assume $Q_1, Q_2 \in \mathbb{S}^Y_s$ with $\mathsf{nest}_\mathsf{v}(Q_1) \leq k$ and $\mathsf{nest}_\mathsf{v}(Q_2) \leq k$. Then $\mathcal{F}[\![Q_1]\!]_k \sqsubseteq_{\mathbb{F}} \mathcal{F}[\![Q_2]\!]_k$ implies $Q_1 \sqsubseteq_{\mathsf{kn}} Q_2$.

Proof. See Appendix. □

Theorem 3.1. For every finite $Y \subseteq \mathcal{N}$ and $s, k \in \mathbb{N}$, $(\mathbb{D}^Y_{s,k}, \sqsubseteq_{\mathsf{kn}})$ is a wqo.

Proof. We have to prove that every infinite sequence of processes $P_1, P_2, \ldots$ in $\mathbb{D}^Y_{s,k}$ is good. Even though $P_1, \cdots \in \mathbb{D}^Y_{s,k}$, we cannot assume that $\mathsf{nest}_\mathsf{v}(P_i) < k$ for every $i \in \mathbb{N}$. But we can have knowledge-congruent processes Q_i for every $i \in \mathbb{N}$ satisfying this restriction: there exists a process $Q_i \in \mathbb{S}_s$ such that $Q_i \equiv_{\mathsf{kn}} P_i$ and $\mathsf{nest}_\mathsf{v}(Q_i) \leq k$. By Lemma 4, this entails that $\mathcal{F}[\![Q_i]\!]_k \in \mathbb{F}^Y_{s,k}$. We know that $(\mathbb{F}^Y_{s,k}, \sqsubseteq_{\mathbb{F}})$ is a wqo and hence the sequence $\mathcal{F}[\![Q_1]\!]_k, \mathcal{F}[\![Q_2]\!]_k, \ldots$ must be a good sequence. This means that there are $i, j \in \mathbb{N}$ with $i < j$, such that $\mathcal{F}[\![Q_i]\!]_k \sqsubseteq_{\mathbb{F}} \mathcal{F}[\![Q_j]\!]_k$. With Lemma 5, we get that $Q_i \sqsubseteq_{\mathsf{kn}} Q_j$. Based on the fact that Q_i is knowledge congruent to P_i as is Q_j to P_j, we know that $P_i \sqsubseteq_{\mathsf{kn}} P_j$. This proves that the sequence $P_1, P_2, \ldots$ is good and the claim follows. □

3.3 Limits and Ideal Decompositions

In the previous section, we built the foundations to provide the finite representations of downward-closed sets by proving the fact that $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$ is a wqo. To this end, we introduce limits as syntactic description of ideals for quasi orders as done in [DS19].

Definition 18 (Ideals)**.** Let $(S, \sqsubseteq)$ be a qo. A set $D \subseteq S$ is an *ideal* if it is downward-closed and directed, i.e. for all $x, y \in D$ there is a $z \in D$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$. We write $\mathrm{Idl}(S)$ for the set of ideals of S.

Single ideals are not sufficient to represent all downward-closed sets. Nevertheless, we can use ideals to represent the latter. This originates in a well-known fact about well-quasi orders: every downward-closed set is equal to a finite union of ideals. In case we require this union to be minimal, it is unique, which is why we can consider a canonical union of ideals which is called *ideal decomposition* of a downward-closed set. Consequently, representing downward-closed sets of $\mathbb{D}_{s,k}$ merely requires us to provide finite representations for its ideals. To this end, we introduce limits as a syntactic construct to denote ideals and define their semantics. They comprise the same syntax as processes and are augmented by -$^\omega$, which is a syntactic construct enabling the denotation of an arbitrary number of parallel components.

Definition 19 (Limits)**.** Limits L are terms that are formed according to the grammar:

$$\begin{aligned}\mathbb{L} \ni L &::= \mathbf{0} \ \mid\ (R_1 \parallel \cdots \parallel R_n)\\ R &::= B \ \mid\ B^\omega\\ B &::= \langle M\rangle \ \mid\ \mathsf{Q}[\vec{a}] \ \mid\ \mathsf{v}x.L\end{aligned}$$

When $n = 0$, $R_1 \parallel \cdots \parallel R_n$ is written $\mathbf{0}$.

Definition 20 (Denotation of Limits)**.** The denotation of L is the set $[\![L]\!] := [\,L\,]{\downarrow}$ where:

$$\begin{aligned}
[\,\mathbf{0}\,] &:= \{\mathbf{0}\} \\
[\,L_1 \parallel L_2\,] &:= \{(P_1 \parallel P_2) \mid P_1 \in [\,L_1\,], P_2 \in [\,L_2\,]\} \\
[\,\langle M\rangle\,] &:= \{\langle M\rangle\} \\
[\,\mathsf{Q}[\vec{a}]\,] &:= \{\mathsf{Q}[\vec{a}]\} \\
[\,\nu x.L\,] &:= \{\nu x.P \mid P \in [\,L\,]\} \\
[\,B^\omega\,] &:= \textstyle\bigcup_{n\in\mathbb{N}} \{(P_1 \parallel \cdots \parallel P_n) \mid \forall i \le n : P_i \in [\,B\,]\}
\end{aligned}$$

The processes in $[\![L]\!]$ are called *instances* of L. We extend the definition of $\mathrm{nest}_\nu(\text{-}) : \mathbb{P} \to \mathbb{N}$ to cover limits by adding the case $\mathrm{nest}_\nu(L^\omega) := \mathrm{nest}_\nu(L)$. It is straightforward to see that for every instance P of a limit L, it holds that $\mathrm{depth}(P) \le \mathrm{nest}_\nu(L)$. Similar to $\mathbb{D}^X_{s,k}$, we define $\mathbb{L}^X_{s,k}$ to be the set of limits L with $\mathrm{nest}_\nu(L) \le k$, free names from X and do not contain messages of size exceeding s. For the same reasons, we may omit X from now on.

Remark 2. Note that every limit is $\sqsubseteq_{\mathsf{kn}}$-downward-closed by definition and hence also closed under $\equiv_{\mathsf{kn}}$. Moreover, iterating a process $\langle M\rangle$ is superfluous due to the persistence of knowledge, i.e. $[\![\langle M\rangle^\omega]\!] = [\![\langle M\rangle]\!]$.

Lemma 6 (Persistence of Knowledge)**.** Let Γ be a set of messages. Then $\langle\Gamma\rangle \parallel \langle\Gamma\rangle \equiv_{\mathsf{kn}} \langle\Gamma\rangle$.

Proof. $\Gamma \le_{\mathsf{kn}} \Gamma, \Gamma$ holds by Axiom (Mon). It remains to show that $\Gamma, \Gamma \le_{\mathsf{kn}} \Gamma$. Let M be any message such that $\Gamma, \Gamma \vdash \Gamma$. Then, it is straightforward to construct a derivability tree (of the same height) for $\Gamma \vdash M$ and hence $\Gamma, \Gamma \le_{\mathsf{kn}} \Gamma$. □

We want to establish the fact that unions of the presented limits are a sound and complete representation of downward-closed sets of $\mathbb{D}_{s,k}$. To this end, we prove that each limit is indeed the denotation of an ideal and vice versa, i.e. $\mathrm{Idl}(\mathbb{D}_{s,k}) = [\![\mathbb{L}_{s,k}]\!]$.

Grounding The denotation of limits are $\sqsubseteq_{\mathsf{kn}}$-downward-closed sets. A natural question to ask is membership of some process in such a set. We introduce means to reduce this question to decidability of $\sqsubseteq_{\mathsf{kn}}$.

Definition 21 (Grounding)**.** The grounding $\lceil\text{-}\rceil^n \colon \mathbb{L}_{s,k} \to \mathbb{D}_{s,k}$ of a limit is defined as follows:

$$\lceil L\rceil^n := \begin{cases} L & \text{if } L \text{ is sequential or } \mathbf{0} \\ \lceil L_1\rceil^n \parallel \lceil L_2\rceil^n & \text{if } L = L_1 \parallel L_2 \\ \nu x.(\lceil L'\rceil^n) & \text{if } L = \nu x.L' \\ (\lceil B\rceil^n)^n & \text{if } L = B^\omega \end{cases}$$

Example 8. Intuitively, grounding replaces every occurrence of ω by some given number n:

$$\begin{aligned}\lceil(\nu x.\langle x\rangle \parallel (\nu y.\mathsf{Q}[x,y])^\omega)^\omega\rceil^2 = ((&\nu x.\langle x\rangle \parallel (\nu y.\mathsf{Q}[x,y]) \parallel (\nu y.\mathsf{Q}[x,y])) \parallel \\ (&\nu x.\langle x\rangle \parallel (\nu y.\mathsf{Q}[x,y]) \parallel (\nu y.\mathsf{Q}[x,y])))\end{aligned}$$

We state some easy properties of grounding in the following lemmas. The proofs are straightforward or direct consequences of the definitions.

Lemma 7. For every $n \in \mathbb{N}$, $\lceil L\rceil^n \in [\,L\,]$.

Lemma 8. For every $n \le m \in \mathbb{N}$, $\lceil L\rceil^n \sqsubseteq_{\mathsf{kn}} \lceil L\rceil^m$.

Lemma 9. For every $P \in [\![L]\!]$ there exists an $n \in \mathbb{N}$ such that $P \sqsubseteq_{\mathsf{kn}} \lceil L \rceil^n$.

Proof. By induction on the structure of L. □

Theorem 3.2. For every $L \in \mathbb{L}_{s,k}$, the set $[\![L]\!]$ is an ideal of $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$.

Proof. For every process $P \in [\,L\,]$, we know that $\mathrm{nest}_{\mathsf{v}}(P) \leq \mathrm{nest}_{\mathsf{v}}(L) \leq k$ which implies that the depth bound k is respected; all messages in P are renamings of messages in L and hence the message sizes never exceed s. These two facts entail that $[\![L]\!] \subseteq \mathbb{D}_{s,k}$. Moreover, $[\![L]\!]$ is downward-closed by definition. It remains to show that $[\![L]\!]$ is directed. Let $P_1, P_2 \in [\![L]\!]$ be two processes. By Lemma 9, we obtain $n_1, n_2 \in \mathbb{N}$ such that $P_i \sqsubseteq_{\mathsf{kn}} \lceil L \rceil^{n_i}$, for $i = 1, 2$. Then, $P_1, P_2 \sqsubseteq_{\mathsf{kn}} \lceil L \rceil^{\max(n_1,n_2)}$. By Lemma 8, we know that $\lceil L \rceil^{\max(n_1,n_2)} \in [\![L]\!]$ and have shown that $[\![L]\!]$ is directed. □

Theorem 3.3. Every ideal in $\mathrm{Idl}(\mathbb{D}_k \cap \mathbb{S}_s)$ is the denotation of some limit $L \in \mathbb{L}_{s,k}$.

Proof. The full proof can be found in [DS19]. We only want to sketch the main ideas. We have introduced multisets in order to apply known facts about ideal completion to the latter. To this end, we apply the forest encoding $\mathcal{F}[\![\text{-}]\!]$ to show that an ideal of $\mathbb{D}_{s,k}$ corresponds to a downward-closed set of $\mathbb{F}_{s,k}$. In order to obtain a limit from these multisets, we use the representations for ideals of multisets introduced in [FG09], i.e. $\circledast$-products. □

3.4 Decidability of Inclusion

We have introduced limits as a mean to finitely represent downward-closed sets. Our next step is to show decidability of inclusion between two such representations of downward-closed sets. Let $D_1, D_2 \subseteq \mathbb{D}_{s,k}$ be two $\sqsubseteq_{\mathsf{kn}}$-downward-closed sets with their ideal decomposition:

$$D_1 = I_1 \cup \ldots \cup I_n \text{ and } D_2 = J_1 \cup \ldots \cup J_m.$$

It is straightforward that $D_1 \subseteq D_2$ holds if and only if for every $1 \leq i \leq n$, $I_i \subseteq D_2$ holds. We can exploit a well-known fact about ideals which is a result of their property of being directed to simplify this check [Frä86].

Lemma 10. Let I, J_1, J_2 be three ideals of a wqo $(D, \leq)$. Then, $I \subseteq J_1 \cup J_2$ iff $I \subseteq J_1$ or $I \subseteq J_2$.

Proof. Towards a contradiction, $I \not\subseteq J_1$ and $I \not\subseteq J_2$. Let $j_1, j_2 \in I$ be two elements such that $j_1 \in J_1 \setminus J_2$ and $j_2 \in J_2 \setminus J_1$. As I is directed, there is a $i \in I$ such that $j_1 \leq i$ and $j_2 \leq i$. We know that $I \subseteq J_1 \cup J_2$ and hence $i \in J_1 \cup J_2$. W.l.o.g. let $i \in J_1$. Ideals are downward-closed, so $j_2 \in J_1$, which is a contradiction. □

Hence, $D_1 \subseteq D_2$ holds iff for every $1 \leq i \leq n$, there is a $1 \leq j \leq m$ such that $I_i \subseteq I_j$. For the decidability of inclusion of two downward-closed sets, this entails that it suffices that inclusion of two ideals is decidable. Single ideals are represented by limits in our domain so that we will develop a constructive proof to compare two limits.

Structural Congruence for Limits For this purpose, we extend structural congruence and introduce a normal form for limits. Structural congruence for limits consists of the same rules as the one for processes, i.e. scope extrusion, α-renaming etc., but we additionally require that $\langle M\rangle^\omega \equiv_{\mathsf{kn}} \langle M\rangle$ for every message M. The latter reflects the intuition of persistence which describes that knowledge that was known once will not be forgotten. We exploit notation by using $\equiv_{\mathsf{kn}}$ for limits as well. It is straightforward to see that $L_1 \equiv_{\mathsf{kn}} L_2$ entails $[\![L_1]\!] = [\![L_2]\!]$.

Remark 3 (Structural congruence and ω's)**.** Note that we did not introduce any means to transfer terms from within an ω to its context. It is possible to define a coarser congruence relation with which we can exploit the semantics of ω. We will discuss such a relation in Section 4.6. It can be used to speed up the implementation of the inclusion check.

Standard Form for Limits Every limit is structurally congruent to a limit of the form $\nu\vec{x}.(\langle\Gamma\rangle \parallel \prod_{i\in I}\mathsf{Q}_i[\vec{M}_i] \parallel \prod_{j\in J} B_j^\omega)$ where every name in $\vec{x}$ occurs free at least once in the scope of the restriction, and for all $j \in J$, B_j is also in standard form. When we write $\mathrm{sf}(L) \overset{\alpha}{=} \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$ we imply that Q is a parallel composition of process calls $\prod_{i\in I}\mathsf{Q}_i[\vec{M}_i]$ (in which case we write $|Q|$ for $|I|$) and R is a parallel composition of iterated limits $\prod_{j\in J} B_j^\omega$. Note that the standard form is in some sense recursive by definition as every B_j is in standard form.

The Absorption Axiom The decidability proof hinges on a characterisation of inclusion that requires an additional hypothesis on the intruder model.

Definition 22 (Absorbing Intruder)**.** Fix an intruder model $\mathbb{I} = (\Sigma, \vdash)$. Let $\vec{x}$ and $\vec{y}$ be two lists of pairwise distinct names, Γ be a finite set of messages, and $\Gamma' = \Gamma[\vec{y}/\vec{x}]$. Moreover, assume that $\mathrm{names}(\Gamma) \cap \vec{y} = \emptyset$. We say $\mathbb{I}$ is *absorbing* if, for all messages M with $\mathrm{names}(M) \subseteq \mathrm{names}(\Gamma)$, we have $\Gamma, \Gamma' \vdash M$ if and only if $\Gamma \vdash M$.

Lemma 11 ([DS19])**.** $\mathbb{I}_{\mathsf{sy}}$ is absorbing.

For the rest of this thesis, we assume an absorbing intruder model.

To understand the idea of the absorption axiom, consider a limit of the form $L = \big(\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)\big)^\omega$. When comparing the difference in knowledge of both groundings $\lceil L\rceil^1$ and $\lceil L\rceil^2$: we have $\mathrm{sf}(\lceil L\rceil^2) \overset{\alpha}{=} \big(\nu\vec{x}.\nu\vec{x}'.(\langle\Gamma\rangle \parallel \langle\Gamma'\rangle \parallel Q \parallel Q')\big)$, where $\Gamma' = \Gamma[\vec{x}'/\vec{x}]$. Imagine we want to check whether a process $\nu\vec{x}.\langle M\rangle$ is embedded in $\lceil L\rceil^2$. As M only contains free names and names in $\vec{x}$, the absorption axiom permits us to ignore Γ' and merely check whether M is in Γ. So basically, it suffices to check whether $\nu\vec{x}.\langle M\rangle \sqsubseteq_{\mathsf{kn}} \lceil L\rceil^1$.

Eventually, we want to prove inclusion of two limits. Let us consider the easier problem of inclusion of a single process in a limit to see how to apply the absorption axiom to our problem. Lemma 9 states that for every process $P \in [\![L]\!]$ for some limit L, there is a n such that $P \sqsubseteq_{\mathsf{kn}} \lceil L\rceil^n$. This easily gives a semi-decision procedure to check whether a process P is contained in some limit L: start with some grounding n of L and increase the factor until P is embedded in the grounding. But in case P is not contained in L, this procedure does not terminate. Intuitively, there should be some bound b that is the maximum factor to ground with so that P could be still included. Every larger grounding $\lceil L\rceil^n$ for $n > b$ is irrelevant for membership of P as we exploit the absorption axiom. Indeed, such a b is given by the number of name restrictions and parallel compositions of process calls. We will prove that every component produced by unfolding a sublimit more than b times does not contribute new knowledge that is relevant for the membership of P.

In contrast to the above example of membership, we deal with a multi-layered limit for the inclusion check. Our algorithm intuitively descends into these layers and it is important to keep track of sublimits we can still use. Hence, we introduce a variant of grounding, called the n-th extension $L \otimes n$, of some limit L (for $n \in \mathbb{N}$).

Definition 23 (Extension)**.** We define extension $- \otimes n \colon \mathbb{L}_{s,k} \to \mathbb{L}_{s,k}$ of a limit as follows:

$$L \otimes n := \begin{cases} L & \text{if } L \text{ is sequential or } \mathbf{0} \\ L_1 \otimes n \parallel L_2 \otimes n & \text{if } L = L_1 \parallel L_2 \\ \nu x.(L' \otimes n) & \text{if } L = \nu x.L' \\ (B \otimes n)^n \parallel B^\omega & \text{if } L = B^\omega \end{cases}$$

Intuitively, extension imitates grounding but keeps all the sublimits B^ω also with an ω. Let us state two basic properties of extension $- \otimes n$.

Lemma 12. For every limit L and $n \in \mathbb{N}$, $[\![L]\!] = [\![L \otimes n]\!]$.

Lemma 13. For every limit L and $n \in \mathbb{N}$, $\lceil L \rceil^n = \lceil L \otimes n \rceil^0$.

Proof. Direct consequence of both definitions. □

Example 9. Note that we cannot generalise Lemma 13: $\lceil L \rceil^{n+m} \neq \lceil L \otimes n \rceil^m$. For instance, consider the following limit:

$$L := (\nu x.(\nu y.\mathsf{Q}[x,y])^\omega)^\omega.$$

If we extend it by 1

$$L \otimes 1 = (\nu x.(\nu y.\mathsf{Q}[x,y])^\omega)^\omega \parallel \nu x.((\nu y.\mathsf{Q}[x,y])^\omega \parallel (\nu y.\mathsf{Q}[x,y]))$$

and then expand the result by 1, we obtain

$$\lceil L \otimes 1 \rceil^1 = (\nu x.(\nu y.\mathsf{Q}[x,y])) \parallel (\nu x.(\nu y.\mathsf{Q}[x,y])^2).$$

In contrast, if we simply expand by 2, we obtain

$$(\nu x.(\nu y.\mathsf{Q}[x,y])^2)^2$$

in which $\nu y.\mathsf{Q}[x,y]$ occurs twice in both parts.

Remark 4. If there is only one level of ω's — which will be denoted by ω-height$(L) = 1$ in terms of definitions that will be introduced in Section 4.6 — then $\lceil L \rceil^{n+m} = \lceil L \otimes n \rceil^m$ for all m and n.

Definition 24 (Source Paths)**.** For a limit L, we extend the definition of the n-th expansion of L by paths indicating from which part of L a subterm of $\lceil L \rceil^n$ stems: We define $\lceil L_\pi \rceil^n$ recursively:

$$\lceil L_\pi \rceil^n := \begin{cases} L_\pi & \text{if } L \text{ is sequential or } \mathbf{0} \\ (\prod_{j=0}^{n-1} \lceil (B_j)_{\pi j} \rceil^n)_\pi & \text{if } L = \prod_{j=0}^{n-1} B_j \\ (\nu x.(\lceil L'_{(\pi 0)} \rceil^n))_\pi & \text{if } L = \nu x.L' \\ \lceil (B_\pi)^n \rceil^n_\pi & \text{if } L = B^\omega \end{cases}$$

where the parallel composition for ω is delegated to the rule of parallel composition. Considering a process $P \in [\![L]\!]$, we know that there is an $n \in \mathbb{N}$ such that $P \sqsubseteq_{\mathsf{kn}} \lceil L \rceil^n$ which entails that $P \equiv \nu\vec{x}.(\langle\Gamma\rangle \parallel Q)$ and $\lceil L\rceil^n \equiv \nu\vec{x}.\nu\vec{x}'.(\langle\Gamma'\rangle \parallel Q \parallel Q')$ with $\Gamma \leq_{\mathsf{kn}} \Gamma'$. By this knowledge embedding, we have a correspondence between names and process calls of P and $\lceil L\rceil^n$ and can hence adopt their paths. This results in a source path annotated process P_π in which only names and process calls are annotated. We may refer to the *position* of a name, process call or process instead of its path in a limit. Due to $\Gamma \leq_{\mathsf{kn}} \Gamma'$, it cannot be guaranteed that paths for messages can be constructed.

It is a also worth noting that a source path annotation is dependent on the limit we consider. The term source path reflects the idea that it is not the path constructed by the structure of the process itself but the limit of origin.

Example 10. Let us consider the source-path annotations for the limit we expanded by 2 in Example 9:

$$\nu x_{00}.((\nu y_{0000}.\mathsf{Q}[x,y]_{00000})_{000} \parallel_{00} (\nu y_{0010}.\mathsf{Q}[x,y]_{00100})_{001})_0 \parallel_\varepsilon$$
$$\nu x_{10}.((\nu y_{1000}.\mathsf{Q}[x,y]_{10000})_{100} \parallel_{10} (\nu y_{1010}.\mathsf{Q}[x,y]_{10100})_{101})_1.$$

We explicitly start with the empty annotation ε and descend recursively according to the definition. Note that the process calls have their own label as they are a single parallel composition with respect to the definition of limits. We also use source path annotations in Example 20.

Definition 25 (Originally Fixed and Non-Fixed parts)**.** Consider a source path annotated process $P_\pi \in [\![L]\!]$. We define every subterm $C_{\pi'}$ to be an *originally fixed part of* L iff π' does not contain any ω in L. Conversely, a subterm $C_{\pi'}$ is an *originally non-fixed part of* L iff π' contains an ω in L. We may also refer to the sets of subterms satisfying these properties and omit the term originally. Note that non-fixed parts is a term for instances of a limit while iterated components correspond to the limit itself. If clear from context, we may use them interchangeably.

Due to the absence of source paths for messages in a process, we cannot classify messages easily, which is not necessary for now. Prior to turning to the main result of this section, we state and prove a property about grounding that is connected to checking inclusion and will be used throughout the proofs.

Lemma 14. Let $L_1, L_2 \in \mathbb{L}$, then $[\![L_1]\!] \subseteq [\![L_2]\!] \iff \forall n \in \mathbb{N}\colon \exists m \in \mathbb{N}\colon \lceil L_1\rceil^n \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$.

Proof. First, let $n \in \mathbb{N}$. We compute the expansion $\lceil L_1\rceil^n \in [\![L_1]\!]$. By assumption, $\lceil L_1\rceil^n \in [\![L_2]\!]$ and by Lemma 9 there exists an m such that $\lceil L_1\rceil^n \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$ as required.
Second, let $P \in [\![L_1]\!]$. We need to show that $P \in [\![L_2]\!]$. By Lemma 9, there exists an expansion for some n with $P \sqsubseteq_{\mathsf{kn}} \lceil L_1\rceil^n$. By assumption, there is an m such that $\lceil L_1\rceil^n \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$. By transitivity of $\sqsubseteq_{\mathsf{kn}}$, we know that $P \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$ which is in $[\![L_2]\!]$. By downward closure, the claim follows. □

Theorem 3.4 (Characterisation of Limits Inclusion)**.** Let L_1 and L_2 be two limits, with $\mathrm{sf}(L_1) \overset{\alpha}{=} \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \prod_{i\in I} B_i^\omega)$, and let $n = |\vec{x}_1| + |Q_1| + 1$. Then:

$$[\![L_1]\!] \subseteq [\![L_2]\!] \iff \begin{cases} \mathrm{sf}(L_2 \otimes n) \overset{\alpha}{=} \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel R_2) \text{ and } \Gamma_1 \leq_{\mathsf{kn}} \Gamma_2 & \textbf{(A)} \\ [\![\langle\Gamma_1\rangle \parallel \prod_{i\in I} B_i]\!] \subseteq [\![\langle\Gamma_2\rangle \parallel R_2]\!] & \textbf{(B)} \end{cases}$$

Theorem 3.5. Given $L_1, L_2 \in \mathbb{L}_{s,k}$ for some $s, k \in \mathbb{N}$, it is decidable whether $[\![L_1]\!] \subseteq [\![L_2]\!]$.

Proof. Based on the characterisation in Theorem 3.4, we design an algorithm recursing on the structure of L_1. One computes the two standard forms $\mathrm{sf}(L_1)$ and $\mathrm{sf}(L_2 \otimes n)$. For every α-renaming that makes condition **(A)** hold, one checks condition **(B)** (recursively). If no α-renaming satisfies both conditions, the inclusion does not hold. In every recursive check of condition **(B)**, there are fewer occurrences of ω in the limit on the left. This is why there is no occurrence of ω in L_1 eventually and it suffices to check condition **(A)**. $\square$

In order to prove Theorem 3.4, we show that the presented conditions are both sufficient and necessary. Let us state some facts about knowledge order $\leq_{\mathsf{kn}}$ and knowledge embedding $\sqsubseteq_{\mathsf{kn}}$. The proofs can be found in the Appendix.

3.4.1 Facts about $\leq_{\mathsf{kn}}$ and $\sqsubseteq_{\mathsf{kn}}$

Lemma 15. Let $\Gamma_1, \Gamma_2, \Gamma$ be sets of messages such that $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$. Then, $\Gamma, \Gamma_1 \leq_{\mathsf{kn}} \Gamma, \Gamma_2$.

Corollary 1. Let $\Delta_1, \Delta_2, \Gamma_1, \Gamma_2$ be sets of messages s.t. $\Delta_1 \leq_{\mathsf{kn}} \Delta_2$ and $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$. Then, $\Delta_1, \Gamma_1 \leq_{\mathsf{kn}} \Delta_2, \Gamma_2$.

Corollary 2. Let Γ be a set of messages, $P_1, P_2 \in \mathbb{P}$ two processes for which $\mathrm{sf}(P_i) = \nu\vec{x}_i.(\langle\Gamma_i\rangle \parallel Q_i)$ for $i \in \{1,2\}$ and $P_1 \sqsubseteq_{\mathsf{kn}} P_2$. Then, $\langle\Gamma\rangle \parallel P_1 \sqsubseteq_{\mathsf{kn}} \langle\Gamma\rangle \parallel P_2$.

Lemma 16. Let P_1, P_2 be two processes and $n \in \mathbb{N}$. If $P_1 \sqsubseteq_{\mathsf{kn}} P_2$, then $P_1^n \sqsubseteq_{\mathsf{kn}} P_2^n$.

Lemma 17. Let $L' \in \mathbb{L}$ a limit s.t. $L' = L^\omega$ for some $L \in \mathbb{L}$. Then, $(\lceil L' \rceil^n)^m \sqsubseteq_{\mathsf{kn}} \lceil L' \rceil^{m*n}$ holds for every $m, n \in \mathbb{N}$.

3.4.2 Sufficient Conditions

Lemma 18 (Sufficient Conditions)**.** If conditions **(A)** and **(B)** hold, then $[\![L_1]\!] \subseteq [\![L_2]\!]$.

Proof. Given that **(A)** and **(B)** hold, it suffices to show that $\forall n_1 \in \mathbb{N}, \exists n_2 \in \mathbb{N}. \lceil L_1 \rceil^{n_1} \sqsubseteq_{\mathsf{kn}} \lceil L_2 \otimes n \rceil^{n_2}$ by Lemmas 12 and 14. Let n_1 be an arbitrary number. With $R_1 := \prod_{i \in I} B_i^\omega$, we have $\lceil \mathrm{sf}(L_1) \rceil^{n_1} = \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \lceil R_1 \rceil^{n_1})$ and $\mathrm{sf}(\lceil L_2 \otimes n \rceil^0) \stackrel{\alpha}{=} \nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2)$ by **(A)**.

We show that $\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \lceil R_1 \rceil^{n_1}) \sqsubseteq_{\mathsf{kn}} \nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel \lceil R_2 \rceil^{n_2})$ for some n_2. Condition **(A)** induces a knowledge order for both fixed parts. Condition **(B)** gives rise to some relation between the iterated components we will exploit. This relation is captured in the following fact. For some α-renaming in **(A)** and some n_2:

$$\langle\Gamma_1\rangle \parallel \lceil R_1 \rceil^{n_1} \sqsubseteq_{\mathsf{kn}} \langle\Gamma_2\rangle \parallel \lceil R_2 \rceil^{n_2} \tag{*}$$

Prior to proving (*), we show that this fact and **(A)** suffice to prove our goal. With $\mathrm{sf}(\lceil R_i \rceil^{n_i}) = \nu\vec{y_i}.(\Gamma_{R_i^{n_i}} \parallel Q_i')$, we call $\Gamma_{R_i^{n_i}}$ the knowledge of $\lceil R_i \rceil^{n_i}$ for $i \in \{1,2\}$. Because of ($*$), we can choose $\vec{y}_2$ so that $\vec{y}_2 = \vec{y}_1, \vec{z}_2$ and there is no need to rename for the embedding anymore. Equipped with this abbreviation, the knowledge of the left hand side $(\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \lceil R_1 \rceil^{n_1})$ is $\Gamma_1, \Gamma_{R_1^{n_1}}$, while $\Gamma_2, \Gamma_{R_2^{n_2}}$ is the knowledge of the right hand side $\nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel \lceil R_2 \rceil^{n_2})$. This is also exactly the knowledge contained in

$(\langle\Gamma_1\rangle \parallel \lceil R_1\rceil^{n_1})$ respectively $(\langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{n_2})$, hence $\Gamma_1, \Gamma_{R_1^{n_1}} \leq_{\mathsf{kn}} \Gamma_2, \Gamma_{R_2^{n_2}}$ by (*). It remains to take care of names and process calls for the embedding. Condition **(A)** already takes care of non-iterated names and process calls. With our assumption (*), we obtain

$$\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \langle\Gamma_1\rangle \parallel \lceil R_1\rceil^{n_1}) \sqsubseteq_{\mathsf{kn}} \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel \langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{n_2}).$$

Both parts are knowledge congruent to our goal's sides by Lemma 6 which proves the goal.

It remains to show that (*) holds, i.e. $(\langle\Gamma_1\rangle \parallel \lceil R_1\rceil^{n_1}) \sqsubseteq_{\mathsf{kn}} (\langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{n_2})$ for some n_2. We start with **(B)** and apply Lemma 14 so that we know that for every $n_1 \in \mathbb{N}$, there is a $m \in \mathbb{N}$ such that $\langle\Gamma_1\rangle \parallel \lceil \prod_{i\in I} B_1\rceil^{n_1} \sqsubseteq_{\mathsf{kn}} \langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{m}$. By Lemma 16, we multiply on both sides:

$$(\langle\Gamma_1\rangle \parallel \lceil \textstyle\prod_{i\in I} B_1\rceil^{n_1})^{n_1} \sqsubseteq_{\mathsf{kn}} (\langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{m})^{n_1}.$$

By Lemma 6, we pull the messages out of both replications:

$$\langle\Gamma_1\rangle \parallel (\lceil \textstyle\prod_{i\in I} B_1\rceil^{n_1})^{n_1} \sqsubseteq_{\mathsf{kn}} \langle\Gamma_2\rangle \parallel (\lceil R_2\rceil^{m})^{n_1}.$$

We continue on both sides individually. On the left, we omit messages for simplicity as we already know that $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$.

$$(\lceil \textstyle\prod_{i\in I} B_1\rceil^{n_1})^{n_1} = (\prod_{i\in I} \lceil B_1\rceil^{n_1})^{n_1} = \lceil R_1\rceil^{n_1}.$$

While we use Lemma 17 and Corollary 2 on the right in order to obtain that:

$$\langle\Gamma_2\rangle \parallel (\lceil R_2\rceil^{m})^{n_1} \sqsubseteq_{\mathsf{kn}} \langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{n_1\cdot m}$$

Combining both paths leads to: $\langle\Gamma_1\rangle \parallel \lceil R_1\rceil^{n_1} \sqsubseteq_{\mathsf{kn}} \langle\Gamma_2\rangle \parallel \lceil R_2\rceil^{n_1\cdot m}$. By choosing $n_2 = n_1 \cdot m$, the claim follows. ☐

3.4.3 Necessary Conditions

Lemma 19 (Necessary Conditions)**.** If $[\![L_1]\!] \subseteq [\![L_2]\!]$, then conditions **(A)** and **(B)** hold.

For this proof we assume the intruder model to be absorbing. Let us first establish some auxiliary definitions and results.

Corollary 3. *Let $\vec{x}$ and $\vec{y}$ be two lists of pairwise distinct names, Γ, Γ_1 be two finite sets of messages, and $\Gamma_2 = \Gamma_1[\vec{y}/\vec{x}]$. Moreover, assume that* $\text{names}(\Gamma_1) \cap \vec{y} = \emptyset$ *and* $\text{names}(\Gamma) \cap \vec{y} = \emptyset = \text{names}(\Gamma) \cap \vec{x}$. *Then, for all messages M with* $\text{names}(M) \subseteq \text{names}(\Gamma, \Gamma_1)$, *we have that* $\Gamma, \Gamma_1, \Gamma_2 \vdash M$ *if and only if* $\Gamma, \Gamma_1 \vdash M$.

Proof. The direction from right to left is obvious. For the reverse direction, it is equivalent to show that $\Gamma, \Gamma_1, \Gamma, \Gamma_2 \vdash M$. Now, $\Gamma, \Gamma_1 = (\Gamma, \Gamma_2)[\vec{y}/\vec{x}]$. The claim follows by the assumption that the intruder is absorbing (Definition 22). ☐

Lemma 20. *Let L, L' be two limits such that $L \equiv L'$. Then for every $n \in \mathbb{N} : \lceil L\rceil^n \equiv \lceil L'\rceil^n$.*

Proof. The proof is a straightforward structural induction on L. ☐

We introduce a refinement of grounding and a function folding the right hand side but preserving the knowledge embedding. The (n, k, m)-th grounding takes a limit and unfolds each ω n times for the outer k nested levels of ω, and m times for the inner ones.

Definition 26 (Step-Indexed Grounding)**.** For a limit L in standard form, we define the (n,k,m)-th grounding of L to be the process $\lceil L\rceil^{n,k,m}$ recursively:

$$\lceil L\rceil^{n,k,m} := \begin{cases} L & \text{if } L \text{ is sequential or } \mathbf{0} \\ \lceil L_1\rceil^{n,k,m} \parallel \lceil L_2\rceil^{n,k,m} & \text{if } L = L_1 \parallel L_2 \\ \nu x.(\lceil L'\rceil^{n,k,m}) & \text{if } L = \nu x.L' \\ \left(\lceil B\rceil^{n,k-1,m}\right)^n & \text{if } L = B^\omega \wedge k > 0 \\ (\lceil B\rceil^m)^m & \text{if } L = B^\omega \wedge k = 0 \end{cases}$$

Definition 27 (ω-height)**.** For a limit L, we define ω-height as follows:

$$\omega\text{-height}(L) := \begin{cases} 0 & \text{if } L \text{ is sequential or } \mathbf{0} \\ \max(\omega\text{-height}(R_1), \cdots, \omega\text{-height}(R_n)) & \text{if } L = R_1 \parallel \cdots \parallel R_n \\ \omega\text{-height}(L') & \text{if } L = \nu x.L' \\ \omega\text{-height}(B) + 1 & \text{if } L = B^\omega \end{cases}$$

Lemma 21. Let L be a limit and $m,n,k \in \mathbb{N}$. If $k \geq \omega\text{-height}(L)$, then $\lceil L\rceil^{n,k,m} = \lceil L\rceil^n$.

Proof. The parameter k only decreases when recursing into a limit under ω. If $k \geq \omega\text{-height}(L)$ the last case of the definition will never apply, which makes the definition coincide with the one of n-grounding. $\square$

The idea of the following parametrised function is to fold an m-grounding to an n-grounding up to a certain ω-height k of the limit. Since we want to be very specific about the domains, we define some sets of groundings.

Definition 28 (Set of Groundings)**.** Let $L \in \mathbb{L}$ and $m,n,k \in \mathbb{N}$.
We define the following two sets of processes:

- $\mathcal{R}_L^m := \{\lceil \mathrm{sf}(L)\rceil^m\}$
- $\mathcal{R}_L^{n,k,m} := \{\lceil \mathrm{sf}(L)\rceil^{n,k,m}\}$

Definition 29. Let L be a limit in recursive standard form and $m,k,n \in \mathbb{N}$. The function $\Phi_{k,L}^{n,m} : \mathcal{R}_L^m \to \mathcal{R}_L^{n,k,m}$ and is parametrised in all four variables.

$$\Phi_{k,P}^{n,m}(P) := \begin{cases} P & \text{if } k = 0 \\ \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel \prod_{j\in J}\left(\Phi_{k,L_j}^{n,m}(\lceil L_j^\omega\rceil^m)\right) & \text{if } k > 0 \text{ and } P = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel \prod_{j\in J} L_j^\omega) \\ \left(\Phi_{k-1,L}^{n,m}(\lceil L\rceil^m)\right)^n & \text{if } k > 0 \text{ and } P = \lceil L^\omega\rceil^m \end{cases}$$

We may omit the parameters n and m in the following if they are obvious from context as they do not change over the process of folding.

Definition 30 (Multi-Hole Contexts)**.** We define multi-hole contexts inductively: the actual base case is a “zero-hole context” which is a plain process as defined in Definition 5. Building on this, we define single-hole contexts $\mathcal{C}_1$ with which we construct multi-hole contexts $\mathcal{C}_n$ for arbitrary n.

$$\mathcal{C}_{n+1} ::= \nu x.\mathcal{C}_{n+1} \mid \mathcal{C}_n \parallel \mathcal{C}_1 \qquad \mathcal{C}_1 ::= [\bullet] \mid \nu x.\mathcal{C}_1 \mid \mathcal{C}_1 \parallel P \mid P \parallel \mathcal{C}_1$$

We can fill an n-ary context by $\mathcal{C}_n[P_1, \cdots, P_n]$ which denotes that $P_1, \cdots, P_n$ are input in preorder into the syntax tree of $\mathcal{C}_n$. When clear from context, we may omit the arity n.

Lemma 22 (Folding is Sound w.r.t. $\sqsubseteq_{\mathsf{kn}}$)**.** Let L_1, L_2 be two limits in standard form $L_i = \nu\vec{x}_i.(\langle\Gamma_i\rangle \parallel Q_i \parallel R_i)$ with $n = |\vec{x_1}| + |Q_1| + 1$. If there is an $m \in \mathbb{N}$ such that $m > n$ and $\lceil L_1\rceil^0 \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$, then $\forall k : \lceil L_1\rceil^0 \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k,L_2}(\lceil L_2\rceil^m)$.

Proof. We prove the claim by induction on k.

For k $= 0$, the claim trivially follows as $\Phi^{n,m}_{0,L_2}(\lceil L_2\rceil^m) = \lceil L_2\rceil^m$ by the assumption that $\lceil L_1\rceil^0 \sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^m$ holds.

For the induction step, we assume that $\lceil L_1\rceil^0 \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k,L_2}(\lceil L_2\rceil^m)$ and prove that

$$\lceil L_1\rceil^0 \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k+1,L_2}(\lceil L_2\rceil^m).$$

By definition of folding, both $\Phi_{k,L}(\lceil L_2\rceil^m)$ and $\Phi_{k+1,L}(\lceil L_2\rceil^m)$ are folding in exactly the same way up to the k-th recursive calls, i.e. calls in which k decreases. This means that up to the calls $\Phi_{0,L'}$ and $\Phi_{1,L'}$, both $\Phi_{k,L}(\lceil L_2\rceil^m)$ and $\Phi_{k+1,L}(\lceil L_2\rceil^m)$ will have constructed the same context $\mathcal{C}[\text{-},\cdots,\text{-}]$ around these final calls. We thus characterise $A = \Phi^{n,m}_{k,L_2}(\lceil L_2\rceil^m)$, and $B = \Phi^{n,m}_{k+1,L_2}(\lceil L_2\rceil^m)$ as follows:

$$\begin{aligned}
A &= \mathcal{C}[\Phi^{n,m}_{0,F_1^\omega}(F_1^\omega),\cdots,\Phi^{n,m}_{0,F_j^\omega}(F_j^\omega)]\\
&= \mathcal{C}[\lceil F_1^\omega\rceil^m,\cdots,\lceil F_j^\omega\rceil^m] = \mathcal{C}[(\lceil F_1\rceil^m)^m,\cdots,(\lceil F_j\rceil^m)^m]\\
B &= \mathcal{C}[\Phi^{n,m}_{1,F_1^\omega}(F_1^\omega),\cdots,\Phi^{n,m}_{1,F_j^\omega}(F_j^\omega)] = \mathcal{C}[(\lceil F_1\rceil^m)^n,\cdots,(\lceil F_j\rceil^m)^n]
\end{aligned}$$

Recall that $L_1 = \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel R_1)$ and hence $\lceil L_1\rceil^0 = \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1)$. By assumption, we have

$$\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1) \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k,L_2}(\lceil L_2\rceil^m) = A \tag{i}$$

and want to prove that

$$\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1) \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k+1,L_2}(\lceil L_2\rceil^m) = B. \tag{ii}$$

Intuitively, we have to find a way to preserve the knowledge embedding from (i) when removing some branches in the holes of the context to get from A to B. Let us show what A and B look like explicitly with their context:

$$\begin{aligned}
A &\equiv \nu\vec{c}.(\langle\Gamma_c\rangle \parallel Q_c \parallel D) &\quad& \text{with } D = (\lceil F_1\rceil^m)^m \parallel \cdots \parallel (\lceil F_j\rceil^m)^m\\
B &\equiv \nu\vec{c}.(\langle\Gamma_c\rangle \parallel Q_c \parallel E) && \text{with } E = (\lceil F_1\rceil^m)^n \parallel \cdots \parallel (\lceil F_j\rceil^m)^n
\end{aligned}$$

We call Γ_c the knowledge of the context.

Our goal is to show that reducing the number of iterations in each of D's parallel components does not affect the knowledge embedding described in (i) and thereby obtain (ii). Let us first consider names and process calls. We can split $\lceil L_1\rceil^0$ in the following way: $\lceil L_1\rceil^0 = \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1) = \nu\vec{y}.\nu\vec{z}.(\langle\Gamma_1\rangle \parallel Q_y \parallel Q_z)$ where $\vec{y} \subseteq \vec{c}$ and $Q_y \subseteq Q_c$. The intention is to distinguish names and process calls that are already matched in the context. This is why we do not require all process calls that are only using names from $\vec{y}$ to be in Q_y but some of them might be in Q_z. The goal follows with the following claim immediately.

Claim I: In every hole of context $\mathcal{C}[\bullet,\cdots,\bullet]$, we can reduce the number of branches to at most n, i.e. for every $1 \leq l \leq j$ we can have a grounding $\lceil F_l\rceil^n$ instead of $\lceil F_j\rceil^m$.

Proof of Claim I. Towards a contradiction, assume that there is a hole in which we cannot remove $m - n$ branches. W.l.o.g. let $\lceil L_l\rceil^m$ for $1 \leq l \leq j$ be the sublimit in this hole. There might be three reasons for this: names, process calls and knowledge.

Considering the names and process calls, we know that $|\vec{x_1}| + |Q_1| \; < \; n$ by definition and hence $|\vec{z}| + |Q_z| \; < \; n$. Now, we investigate the components that $\vec{z}$ and Q_z are matched to. In the worst case, all of them are mapped to this hole but we still delete $m - n$ branches that are not used to match the names $\vec{z}$. For the process calls in Q_z, we have to distinguish two cases. First, if a process call uses any name from $\vec{z}$, it is fine as we will leave them anyway. Second, if a process call does not use any name from $\vec{z}$, it is fine to delete this branch as there will be enough copies in the remaining branches to cover this process call.

It remains to reason about knowledge. We make the knowledge of D, i.e. the one having budget k, explicit: $\mathrm{sf}(D) = \nu\vec{a}.(\Gamma_m \parallel \cdots)$ and factor out the knowledge from sublimit $\lceil F_l \rceil^m$: $\Gamma_m = \Gamma'_m, \Gamma_{l,m}$ so that $\Gamma_{l,m}$ was the knowledge obtained through $\lceil F_l \rceil^m$. For knowledge, we have to prove that $\forall M \colon \Gamma_1 \vdash M \implies \Gamma_c, \Gamma'_m, \Gamma_{l,n} \vdash M$ given that $\Gamma_c, \Gamma'_m, \Gamma_{l,m} \vdash M$. The idea now is to reduce the knowledge from $\Gamma_{l,m}$ to $\Gamma_{l,n}$ by Corollary 3, which is a corollary of the absorbing intruder. Let us define $\Gamma'_{c,m} = \Gamma_c, \Gamma'_m$ indicating the context of the hole we are considering. As m might be bigger than $2n$, we have to iterate the process of reducing the number of branches. Hence, we generalise the notation of $\Gamma_{l,m}$ and $\Gamma_{l,n}$ in the following way: $(\lceil F_l \rceil^m)^i \equiv \nu\vec{a_i}.(\Gamma_{l,i} \parallel \cdots)$.

Claim II: $\forall m > n,\ \Gamma_{l,m} \vdash M \implies \Gamma_{l,m-1} \vdash M$.

Proof of Claim II. For convenience, we rename $\Gamma_{l,i}$ to Λ_i. The main observation is that we can split the knowledge Λ_m into Λ_{m-1} and a remainder Λ'. We can choose a branch which does not use names from $\vec{z}$ to contribute to Λ'. Since we know that $n \geq 1$, we know that $m > 1$ by assumption. Therefore, we can split Λ_{m-1} again and obtain the knowledge stemming from one branch, which we call Λ''. Let us recall the assumption and goal after these rewriting steps: Given that

$$\Gamma'_{c,m}, \Lambda_{m-2}, \Lambda', \Lambda'' \vdash M \tag{iii}$$

holds, we want to prove that $\Gamma'_{c,m}, \Lambda_{m-2}, \Lambda'' \vdash M$. Let $\vec{w}'$ and $\vec{w}''$ be the names only used in Λ' and Λ'' respectively so that:

$$\vec{w}' \cap \vec{w}'' = \emptyset \quad \text{and} \quad \mathrm{names}(\Gamma'_{c,m}, \Lambda_{m-2}) \cap \vec{w}' = \emptyset = \mathrm{names}(\Gamma'_{c,m}, \Lambda_{m-2}) \cap \vec{w}'' \tag{iv}$$

Λ' and Λ'' have been obtained from a branch of the same sublimit, so we can infer that

$$\Lambda'' = \Lambda'[\vec{w}''/\vec{w}']. \tag{v}$$

Notice that $\vec{w}' \cap \vec{z} = \emptyset$ by the fact how we have chosen the branch for Λ'. Furthermore, $\vec{w}' \cap \vec{y} = \emptyset$ by $\vec{y} \subseteq \vec{c}$. Combining these observations, we get that $\vec{x_1} \cap \vec{w}' = \emptyset$. By the Locality-Axiom and $\Gamma_1 \vdash M$, we get $\mathrm{names}(M) \subseteq \mathrm{names}(\Gamma_1) \subseteq \vec{x_1}$. Therefore, $\mathrm{names}(M) \cap \vec{w'} = \emptyset$ which implies that

$$\mathrm{names}(M) \subseteq \mathrm{names}(\Gamma'_{c,m}, \Lambda_{m-2}, \Lambda'') \tag{vi}$$

The facts *(iii)* to *(vi)* fulfil the conditions for Corollary 3 resulting in $\Gamma'_{c,m}, \Lambda_{m-2}, \Lambda'' \vdash M$ which reads $\Gamma'_{c,m}, \Lambda_{m-1} \vdash M$ when folding back which is the goal of Claim II. By this, we have shown that we can remove $m - n$ branches which leads to a contradiction which is why Claim I holds. As $\lceil F_l \rceil^m$ was chosen arbitrarily, we have shown that we can remove $m - n$ branches in every hole of the context $\mathcal{C}[\bullet, \cdots, \bullet]$ which concludes this proof. □

Corollary 4. Let L_1, L_2 be two limits with $\mathrm{sf}(L_1) = \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel R_1)$ and $[\![L_1]\!] \subseteq [\![L_2]\!]$. Then, $\mathrm{sf}(L_2 \otimes n) \overset{\alpha}{=} \nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel R_2)$ and $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ for $n = |\vec{x}_1| + |Q_1| + 1$.

Proof. First, we show that $\lceil \mathrm{sf}(L_1) \rceil^0 \sqsubseteq_{\mathsf{kn}} \lceil L_2 \rceil^n$. By $[\![L_1]\!] \subseteq [\![L_2]\!]$, we know that there is an m so that $\lceil \mathrm{sf}(L_1) \rceil^0 \sqsubseteq_{\mathsf{kn}} \lceil L_2 \rceil^m$ by Lemma 14. From Lemma 22, we obtain that

that $\lceil \mathrm{sf}(L_1)\rceil^0 \sqsubseteq_{\mathsf{kn}} \Phi^{n,m}_{k,L_2}(\lceil L_2\rceil^m)$ for every k. Substituting ω-height(L_2) for k leads to $\Phi^{n,m}_{\omega\text{-height},L_2}(\lceil L_2\rceil^m)$. This is $\lceil L_2\rceil^n$ by Lemma 21. Using this knowledge embedding, we get $\lceil L_2\rceil^n \stackrel{\alpha}{=} \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2)$ with $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$. With Lemma 13, we observe that $\lceil L_2\rceil^n = \lceil L_2 \otimes n\rceil^0$. By this, the claim follows as $\lceil\text{-}\rceil^0$ merely omits the iterated parts, i.e. R_2, from $\mathrm{sf}(L_2 \otimes n)$. □

Lemma 23 (Necessary Conditions for Inclusion)**.** Let L_1 and L_2 be two limits, with $\mathrm{sf}(L_1) \stackrel{\alpha}{=} \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel R_1)$ with $R_1 = \prod_{i\in I} B_i^\omega$, and let $n = |\vec{x}_1|+|Q_1|+1$. Given that the inclusion $[\![L_1]\!] \subseteq [\![L_2]\!]$, both conditions **(A)** and **(B)** hold.

Proof. For **(A)**, the claim follows by Corollary 4 which implicitly gives a renaming for L_2. For **(B)**, we want to show that $[\![\langle\Gamma_1\rangle \parallel \prod_{i\in I} B_i]\!] \subseteq [\![\langle\Gamma_2\rangle \parallel R_2]\!]$.
Let $N_i = \langle\Gamma_i\rangle \parallel R_i$. It is straightforward to see that $[\![\langle\Gamma_1\rangle \parallel \prod_{i\in I} B]\!] \subseteq [\![N_1]\!]$. This is why it is enough to show that $[\![N_1]\!] \subseteq [\![N_2]\!]$ by transitivity.
Towards a contradiction, we assume that for all possible renamings so that **(A)** is satisfied, $[\![N_1]\!] \not\subseteq [\![N_2]\!]$. By Lemma 14,

$$\exists m_1', \forall m_1 \geq m_1', \forall m_2. \lceil N_1\rceil^{m_1} \not\sqsubseteq_{\mathsf{kn}} \lceil N_2\rceil^{m_2}. \qquad (*)$$

First, knowledge could break the embedding. Let us define

$$\mathrm{sf}(\lceil N_i\rceil^{m_i}) = \nu y_i.(\Gamma_i \parallel \Gamma_i' \parallel Q_i').$$

Notice that names$(\Gamma_i) \cap \vec{y_i} = \emptyset$. There might be two reasons why the knowledge embedding does not hold.

First, the embedding breaks because of knowledge. This is impossible as Γ_1, Γ_1' and Γ_2, Γ_2' represent the knowledge of groundings of the two limits L_1 and L_2 as their top level knowledge is replicated in **(B)**. Therefore, the inclusion $[\![L_1]\!] \subseteq [\![L_2]\!]$ would also break which is a contradiction.

Second, names or process calls can hence be the only reasons why the knowledge embedding $\lceil N_1\rceil^{m_1'} \sqsubseteq_{\mathsf{kn}} \lceil N_2\rceil^{m_2}$ does not hold for any m_2. We will derive a contradiction by choosing $n_1 = 2\cdot max(|\vec{x}_2|+|Q_2|+1, m_1')$. We incorporate $\lceil R_1\rceil^{n_1}$ into $\mathrm{sf}(L_1)$: $\lceil\mathrm{sf}(L_1)\rceil^{n_1} = \nu\vec{x}_1.(\Gamma_1 \parallel Q_1 \parallel \lceil R_1\rceil^{n_1})$. Recall that $\lceil\mathrm{sf}(L_2 \otimes n)\rceil^{m_2} \stackrel{\alpha}{=} \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel \lceil R_2\rceil^{m_2})$. By the size of n_1, at least half of the names and process calls of $\lceil R_1\rceil^{n_1}$ have to be covered by $\lceil R_2\rceil^{m_2}$ as the non-iterated part $\vec{x_2}$ and Q_2 cannot do more than half. But by definition $\frac{n_1}{2} \geq m_1'$ so $\forall m_2, \lceil R_1\rceil^{\frac{n_1}{2}} \not\sqsubseteq_{\mathsf{kn}} \lceil R_2\rceil^{m_2}$ as knowledge cannot be the reason for $(*)$ to break. Altogether, this entails that there is an n_1 such that for all m_2:

$$\begin{aligned} \lceil\mathrm{sf}(L_1)\rceil^{n_1} = \quad & \nu\vec{x}_1.(\Gamma_1 \parallel Q_1 \parallel \lceil R_1\rceil^{n_1}) \\ & \qquad\qquad \not\sqsubseteq_{\mathsf{kn}} \\ & \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel \lceil R_2\rceil^{m_2}) \stackrel{\alpha}{=} \lceil\mathrm{sf}(L_2\otimes n)\rceil^{m_2}. \end{aligned}$$

Using Lemma 12, this implies that $\lceil L_1\rceil^{n_1} \not\sqsubseteq_{\mathsf{kn}} \lceil L_2\rceil^{m_2}$ for every m_2. In turn, this entails that $[\![L_1]\!] \not\subseteq [\![L_2]\!]$ by Lemma 14 which is a contradiction. □

3.5 Computing Post-Hat

To instantiate the ideal completions framework, it remains to show that $\widehat{\text{post}}^s_\Delta(L)$ is decidable for any limit L. To be precise, we need to define an algorithm that returns a set of limits $\{L_1, \cdots, L_n\}$ such that $\widehat{\text{post}}^s_\Delta(L) = [\![L_1]\!] \cup \cdots \cup [\![L_2]\!]$ without enumerating all instances of L. Assuming an absorbing intruder, we will be able to show that it suffices to consider some finite expansion of L to exhibit every possible successor, e.g. by factor b. Every unfolding that is produced by some bigger factor will not contribute successors that have not been considered before. Intuitively, this is a consequence of the fact that an input action can only bind a finite number of names due to the message size restrictions. There are two types of transitions to consider. First, for τ-transitions it suffices to expand by 1 as no names are bound in an input pattern and hence occuring in the continuation. Second, for input patterns of the form $\mathbf{in}(\vec{x} : M)$, we want to guarantee that every possible message that can be bound to x can be filled with distinct names stemming from the same position in the limit. Hence, the bound b is not fixed but parametric in the arity of message constructors of the intruder model $\mathbb{I}$ and the patterns of process call definitions Δ. For instance, consider the symmetric intruder model. Given a size bound s on messages, we know that a message M can have at most 2^s distinct names. Recall that the fragment we are considering only requires the processes to respect the message size bound: $\mathbb{D}^X_{s,k} := \{P \in \mathbb{S}_s \mid \text{fn}(P) \subseteq X, \exists Q \in \mathbb{S}_s \colon Q \equiv_{\mathsf{kn}} P \wedge \text{nest}_{\mathsf{v}}(Q) \leq k\}$. So we do not require the message for the pattern match to respect the size bound s. Therefore, we might need 2^s for every variable to be bound when considering the symmetric intruder model.

Definition 31 ($\beta(\Delta)$ and $\gamma(\mathbb{I})$)**.** Let Δ be a set of definitions and $\mathbb{I}$ be an intruder model. We define $\beta(\Delta)$ and $\gamma(\mathbb{I})$ as follows:

$$\begin{aligned} \beta(\Delta) &:= \max\{\, |\vec{x}| \mid (\mathsf{Q}[\vec{y}] := A + \mathbf{in}(\vec{x} : M).P + A') \in \Delta \,\} \\ \gamma(\mathbb{I}) &:= \max\{\, \text{ar}(\mathsf{f}) \mid \mathsf{f} \in \Sigma) \,\} \text{ with } \mathbb{I} = (\Sigma, \vdash). \end{aligned}$$

Definition 32 ($\widehat{\text{post}}$)**.** Let $b = \beta(\Delta) \cdot \gamma(\mathbb{I})^{s-1} + 1$ and $\text{sf}(L \otimes b) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$, then

$$\widehat{\text{post}}^s_\Delta(L) := \left\{\ \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q' \parallel R\big)\ \middle|\ \nu\vec{x}.\big(\langle\Gamma\rangle \parallel Q\big) \to_\Delta \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q'\big) \in \mathbb{S}_s\ \right\}$$

We basically want to prove that every successor of every instance of a limit L is covered by this definition. For this purpose, the following observation enables us to only consider expansions of L instead of arbitrary instances.

Corollary 5 (Post of Expansion is Enough)**.** Let L be a limit and $P \in [\![L]\!]$. Then, for every $P_1 \in \text{post}(P)$, there is a $P_2 \in \text{post}(\lceil L\rceil^n)$ for some $n \in \mathbb{N}$ such that $P_1 \sqsubseteq_{\mathsf{kn}} P_2$.

Proof. As $P \in [\![L]\!]$, we know that there is a $n' \in \mathbb{N}$ such that $P \sqsubseteq_{\mathsf{kn}} \lceil L\rceil^{n'}$ by Lemma 9. We choose this n' to be n. We have that $P \to P_1$ and $P \sqsubseteq_{\mathsf{kn}} \lceil L\rceil^n$. As $\sqsubseteq_{\mathsf{kn}}$ is a simulation (Theorem 2.1), we know that there is a P_2 so that $\lceil L\rceil^n \to P_2$ and $P_1 \sqsubseteq_{\mathsf{kn}} P_2$. □

Theorem 3.6. $\widehat{\text{post}}^s_\Delta(L) = \{L_1, \ldots, L_n\} \implies \text{post}^s_\Delta([\![L]\!])\!\downarrow = [\![L_1]\!] \cup \ldots \cup [\![L_n]\!]$.

Proof. We assume that $\widehat{\text{post}}^s_\Delta(L) = \{L_1, \ldots, L_n\}$ and prove the set equality by two inclusions. First, we show that $\big(\text{post}([\![L]\!]) \cap \mathbb{S}_s\big)\!\downarrow \supseteq [\![L_1]\!] \cup \ldots \cup [\![L_n]\!]$. It suffices to show that $[\![L_j]\!] \subseteq \big(\text{post}([\![L]\!]) \cap \mathbb{S}_s\big)\!\downarrow$ for every $j \in \{1, \cdots, n\}$. We choose j arbitrarily and prove the latter statement. By definition, we know that L_j stems from at least one transition in the non-iterated part of $\text{sf}(L \otimes b) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$. W.l.o.g. let $P_1 := \nu\vec{x}.\big(\langle\Gamma\rangle \parallel Q\big) \to_\Delta$

$\nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q'\big) =: P_2$ be this transition in $\mathbb{S}_s$. Hence, $L_j = \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q' \parallel R\big)$. By Lemma 12, $[\![L]\!] = [\![L \otimes b]\!]$ and therefore $\lceil \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)\rceil^n = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel \lceil R\rceil^n) \in [\![L]\!]$ for every n. Because of $P_1 \to P_2$, we know that $\lceil \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)\rceil^n \to \lceil \nu\vec{y}.(\langle\Gamma'\rangle \parallel Q' \parallel R)\rceil^n$. This is why $\lceil L_j\rceil^n = \lceil \nu\vec{y}.(\langle\Gamma'\rangle \parallel Q' \parallel R)\rceil^n \in \mathrm{post}([\![L]\!])$ for every n. With Lemma 9, we know that for every $P \in [\![L_j]\!]$, there is an m so that $P \sqsubseteq_{\mathsf{kn}} \lceil L_j\rceil^m$. By downward-closure, we infer that $P \in \big([\![\mathrm{post}(L)]\!] \cap \mathbb{S}_s\big)\!\downarrow$.

Second, we show that $\big(\mathrm{post}([\![L]\!]) \cap \mathbb{S}_s\big)\!\downarrow \subseteq [\![L_1]\!] \cup \ldots \cup [\![L_n]\!]$. Let $\lceil L\rceil^b$ has standard form: $\mathrm{sf}(\lceil L\rceil^b) \overset{\alpha}{=} \nu\vec{x}_1.(\Gamma_1 \parallel \mathsf{Q}[\vec{M}] \parallel C_1)$. By Corollary 5, it suffices to consider only successors of groundings of L. Therefore, let $b < m \in \mathbb{N}$ with $\mathrm{sf}(\lceil L\rceil^m) \overset{\alpha}{=} \nu\vec{x}_1,\vec{x}_2.(\langle\Gamma_1\rangle \parallel \langle\Gamma_2\rangle \parallel \mathsf{Q}[\vec{M}] \parallel C_2)$ and $\mathsf{Q}[\vec{M}] := \mathbf{in}(\vec{p} : N).P_1$. W.l.o.g., we derive the message which is matched using some fresh intruder names $\vec{c}$: $\Gamma_1, \vec{c} \vdash N[\vec{M}'/\vec{x}]$. Overall, we obtain the following transition:

$$\lceil L\rceil^m \xrightarrow[\vec{p}\,\to\,\vec{M}']{\mathsf{Q}[\vec{M}]=\mathbf{in}(\vec{p}:N).P_1+A} \nu\vec{x}_1,\vec{x}_2,\vec{c}.(\langle\Gamma_1\rangle \parallel \langle\Gamma_2\rangle \parallel \langle\vec{c}\rangle \parallel P_1[\vec{M}'/\vec{p}] \parallel C_1) =: Q' \in \mathrm{post}^s(L)$$

where the annotations explicitly state which process call and action was used with which substitution. We want to show that we can have the same reduction in $\lceil L\rceil^b$. We do so by proving that this transition is still enabled.

Claim I: In every hole of context $\mathcal{C}[\bullet, \cdots, \bullet]$, we can reduce the number of branches to at most n, i.e. for every j we can have a grounding $\lceil F_j\rceil^n$ instead of $\lceil F_j\rceil^m$, for every $k \in \mathbb{N}$:

$$\exists \vec{y}_k, \Delta_k, D_k \colon \Phi^{b,n}_{k,L}(\lceil L\rceil^n) \xrightarrow[\vec{p}\,\to\,\vec{M}']{\mathsf{Q}[\vec{M}]=\mathbf{in}(\vec{p}:N).P_1+A} \nu\vec{y}_k, \vec{c}.(\langle\Delta_k\rangle \parallel \langle\vec{c}\rangle \parallel P_1[\vec{M}'/\vec{p}] \parallel D_k)$$
$$\text{with } \Delta, \vec{c} \vdash N[\vec{M}'/\vec{p}] \qquad (*)$$

Proof of Claim I by induction on k: For the base case in which $k = 0$, the claim holds by assumption. For the induction step, we assume that $(*)$ holds for k and we prove it for $k+1$. Similar to the proof of Lemma 22, we use a multi-hole context $\mathcal{C}[\bullet, \cdots, \bullet]$ to distinguish between having budget k or $k+1$. Let $F_1, \cdots, F_j$ be j limits and $\mathcal{C}[\bullet, \cdots, \bullet]$ a multi-hole context so that:

$$\begin{aligned}
\Phi^{b,n}_{k,L}(\lceil L\rceil^n) &\equiv \mathcal{C}[\Phi^{b,n}_{0,F_1^\omega}(F_1^\omega), \cdots, \Phi^{b,n}_{0,F_j^\omega}(F_j^\omega)] = \mathcal{C}[\lceil F_1^\omega\rceil^n, \cdots, \lceil F_j^\omega\rceil^n] \\
&= \mathcal{C}[(\lceil F_1\rceil^n)^n, \cdots, (\lceil F_j\rceil^n)^n] = A \\
\Phi^{b,n}_{k+1,L}(\lceil L\rceil^n) &\equiv \mathcal{C}[\Phi^{b,n}_{1,F_1^\omega}(F_1^\omega), \cdots, \Phi^{b,n}_{1,F_j^\omega}(F_j^\omega)] \\
&= \mathcal{C}[(\lceil F_1\rceil^n)^b, \cdots, (\lceil F_j\rceil^n)^b] = B
\end{aligned}$$

We want to show that it suffices to have b copies of F_l for any $1 \le l \le j$. Since $\mathsf{Q}[\vec{M}]$ also occurs in $\mathrm{sf}(\lceil L\rceil^b)$, it is trivial to keep the process call which is reduced. It remains to argue that the same redex is enabled in B. Towards a contradiction: Assume that there is a hole in which b copies are not sufficient. W.l.o.g. let L_l be the limit in this hole.

We did not explicitly state the message $N[\vec{M}'/\vec{p}]$ but the substitution $\vec{p} \to \vec{M}'$ for the continuation since the substitution is solely determining the successor.

We know that A's knowledge is Δ_k and factor out the knowledge from sublimit $\lceil F_l\rceil^m$: $\Delta_k = \Delta'_k, \Delta_{l,m}$ so that $\Delta_{l,m}$ was the knowledge obtained through $\lceil F_l\rceil^m$. We want to prove that $\Delta'_k, \Delta_{l,m}, \vec{c} \vdash N[\vec{M}'/\vec{p}] \implies \Delta'_k, \Delta_{l,b}, \vec{c} \vdash N[\vec{M}'/\vec{p}]$ where $\Delta_{l,b}$ denotes the knowledge obtained through $\lceil F_l\rceil^b$ respectively.

Now, we consider the different names used in $N[\vec{M}'/\vec{p}]$. It is straightforward to see that $\mathrm{names}(N) \subseteq \vec{x}_1$. Recall the definition of b:

$$b := \beta(\Delta) \cdot \gamma(\mathbb{I})^{s-1} + 1$$

Hence, we know that $\beta(\Delta) \geq |\vec{p}|$ and therefore $b \geq |\vec{p}| \cdot \gamma(\mathbb{I})^{s-1} + 1$. Intuitively, this ensures that there are enough distinct names for every single parameter in the parameter list as the size determines the maximum depth of the syntax tree of a message. As names$(\vec{p}) < b$, we can remove at least $m - b$ branches without loosing names used in $\vec{p}$. Therefore, we can assume that names$(N[\vec{M}/\vec{p}]) \in \vec{x}_1$. The idea is to reduce the knowledge from $\Delta_{l,m}$ to $\Delta_{l,b}$ by Corollary 3, which is a corollary of the absorbing intruder. As m might be bigger than $2b$, we have to iterate the process of reducing the number of branches. Hence, we generalise the notation of $\Delta_{l,m}$ and $\Delta_{l,b}$ in the obvious way: $(\lceil F_l \rceil^m)^i \equiv \nu\vec{a_i}.(\Delta_{l,i} \parallel \cdots)$.

Claim II: $\forall m > b$, $\Delta_{l,m} \vdash N[\vec{M}/\vec{p}] \implies \Delta_{l,m-1} \vdash N[\vec{M}/\vec{p}]$.

Proof of Claim II. For convenience, we rename $\Delta_{l,i}$ to Λ_i. The main observation is that we can split the knowledge Λ_m into Λ_{m-1} and a remainder Λ'. We can choose a branch which does not use names from $\vec{x}_1$ to contribute to Λ'. Since we know that $n \geq 1$, we know that $m > 1$ by assumption. Therefore, we can split Λ_{m-1} again and obtain the knowledge stemming from one branch which we call Λ''. Let us recall the assumption and goal after these rewriting steps: Given that

$$\Delta'_k, \Lambda_{m-2}, \Lambda', \Lambda'' \vdash N[\vec{M}/\vec{p}] \tag{iii}$$

holds, we want to prove that $\Delta'_k, \Lambda_{m-2}, \Lambda'' \vdash N[\vec{M}/\vec{p}]$. Let $\vec{w}'$ and $\vec{w}''$ be the names only used in Λ' and Λ'' respectively so that:

$$\vec{w}' \cap \vec{w}'' = \emptyset \text{ and } \operatorname{names}(\Gamma'_{c,m}, \Lambda_{m-2}) \cap \vec{w}' = \emptyset = \operatorname{names}(\Gamma'_{c,m}, \Lambda_{m-2}) \cap \vec{w}'' \tag{iv}$$

Λ' and Λ'' have been obtained from a branch of the same sublimit, so we can infer that

$$\Lambda'' = \Lambda'[\vec{w}''/\vec{w}']. \tag{v}$$

Notice that $\vec{w}' \cap \vec{x}_1 = \emptyset$ by the fact how we have chosen the branch for Λ'. Because of names$(N[\vec{M}/\vec{p}]) \subseteq \vec{x}_1$, we can infer that names$(N[\vec{M}/\vec{p}]) \cap \vec{w}' = \emptyset$ which implies that

$$\operatorname{names}(N[\vec{M}/\vec{p}]) \subseteq \operatorname{names}(\Delta'_k, \Lambda_{m-2}, \Lambda'') \tag{vi}$$

The facts (iii) to (vi) satisfy the conditions for Corollary 3. Applying this corollary leads to $\Delta'_k, \Lambda_{m-2}, \Lambda'' \vdash N[\vec{M}/\vec{p}]$ which reads $\Delta'_k, \Lambda_{m-1} \vdash N[\vec{M}/\vec{p}]$ when folding back which is the goal of Claim II. In turn, this concludes the proof of Claim I.

Instantiating the statement of Claim I with $k > \omega$-height(L) shows that this transition is still enabled in $\lceil L \rceil^b$. Consider the extension $L \otimes b$ of limit L whose standard form we choose to resemble the correlation with $\lceil L \rceil^b$: $\operatorname{sf}(L \otimes b) \overset{\alpha}{=} \nu\vec{x}_1.(\Gamma_1 \parallel \mathsf{Q}[\vec{M}] \parallel C_1 \parallel R_1)$. By the definition of $\widehat{\operatorname{post}}(L)$, we know that

$$L' := \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel P_1[\vec{M}/\vec{p}] \parallel C_1 \parallel R_1) \in \widehat{\operatorname{post}}(L).$$

Recall that $Q' = \nu\vec{x}_1, \vec{x}_2, \vec{c}.(\langle\Gamma_1\rangle \parallel \langle\Gamma_2\rangle \parallel \langle\vec{c}\rangle \parallel P_1[\vec{M}'/\vec{p}] \parallel C_1)$ was the successor of $\lceil L \rceil^m$ from the beginning. It remains to show that $Q' \in [\![L']\!]$. As before for $L \otimes b$, we relate the standard form of $L \otimes m$ to $\lceil L \rceil^m$ and get the following:

$$\operatorname{sf}(L \otimes m) \overset{\alpha}{=} \nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_1\rangle \parallel \langle\Gamma_2\rangle \parallel \mathsf{Q}[\vec{M}] \parallel C_2 \parallel R_2)$$

By Lemma 12, we have that $[\![L \otimes m]\!] = [\![L \otimes b]\!]$ and taking one step hence leads to

$$\begin{array}{rcll} K' & := & [\![\nu\vec{x}_1, \vec{x}_2, \vec{c}.(\langle\Gamma_1\rangle \parallel \langle\Gamma_2\rangle \parallel \langle\vec{c}\rangle \parallel P_1[\vec{M}'/\vec{p}] \parallel C_2 \parallel R_2]\!] & \\ & = & [\![\nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel \langle\vec{c}\rangle \parallel P_1[\vec{M}'/\vec{p}] \parallel C_1 \parallel R_1)]\!] & =: L' \end{array}$$

As $Q' \in K'$, it also holds that $Q' \in L'$ which concludes this proof.

□

3.6 Invariant for Example 2

Equipped with the theory established in this chapter, we want to give an invariant for Example 2. On the one hand, this proves that the protocol is depth-bounded as we are able to give an invariant. On the other hand, it over-approximates the reachable state space and proves secrecy of the session key k as $\mathsf{Leak}[k]$ is not present in the invariant.

The invariant can actually be given as a single limit which is also a property we will comment on in Section 4.1. The following limit L is an inductive invariant for Example 2 when restricting messages in processes to size 3:

$$\begin{aligned}
L &= \nu a, b, k_{as}, k_{bs}.(\langle a, b\rangle \parallel \mathsf{A}_1[a, b, k_{as}] \parallel \mathsf{B}_1[a, b, k_{bs}] \parallel \mathsf{S}[a, b, k_{as}, k_{bs}]^\omega \parallel L_1^\omega) \\
L_1 &= \nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2^\omega\big) \\
L_2 &= \nu k.\big(\langle \mathsf{e}(k)_{(a,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{(b,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{(n_a,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{k_{bs}}\rangle \parallel \mathsf{Secret}[k]^\omega \parallel \\
&\qquad \mathsf{A}_3[a, b, k_{as}, k]^\omega \parallel L_3^\omega\big) \\
L_3 &= \nu n_b.\big(\langle \mathsf{e}(n_b)_{(k,k)}\rangle \parallel \langle \mathsf{e}(n_b)_k\rangle \parallel \mathsf{B}_2[a, b, k_{bs}, n_b, k]\big)
\end{aligned}$$

Since information like the property that k is a secret is uncountable, it might look artificial that $\mathsf{Secret}[k]$ is decorated with an ω. This is a modelling artefact as we use process calls to model properties like secrecy. In contrast to messages, process calls can be consumed and hence need to decorate $\mathsf{Secret}[k]$ with an ω in the inductive invariant.

The number of name restrictions is finite: $\mathrm{nest}_\nu(L) = 7$. Hence, the invariant certifies that any process of $[\![L]\!]$ is $(3, 7)$-bounded. It also satisfies secrecy of k as $\mathsf{Leak}[k]$ does not occur. Since $P_0 \in [\![L]\!]$, we also know that all processes reachable from P_0 satisfy these properties. In fact, L^ω is also an inductive invariant and therefore the same properties hold for P_0^ω. Moreover, we obtain that $[\![\nu k.(\nu x.(\langle(x, \mathsf{e}(x)_k)\rangle)^\omega\rangle]\!] \not\subseteq [\![L^\omega]\!]$. Hence, the invariant L^ω proves that the protocol is not susceptible to known-/chosen-plaintext attacks.

We present the benchmark results for Example 2 obtained when using our prototype implementation in Chapter 5. The limit L was automatically inferred and the results also show that the limit(s) are indeed inductive.

Chapter 4

Algorithmic Aspects

The previous chapters established rather theoretical decidability results. We now turn to computability results: we will explain how to apply these foundations in practice and present observations to lower the computational workload. To start with, we explain how we can exploit the relation of two limits when checking inclusion: $\widehat{\text{post}}(L) \subseteq L$ for some L; since our characterisation did not account for this. While this check is capable to handle the generic intruder model, we will then turn to the intruder model for symmetric encryption and present some results to handle knowledge algorithmically. This will lead to a method to handle the pattern matching for the symmetric intruder model, which should be adoptable to generic intruder models. Equipped with the means to do the pattern matching, we present a generic way of generating invariants and exemplify it. Then, we present how to encode knowledge embedding as an SMT instance. Lastly, we generalise the notion of structural congruence for limits and present an algorithm with which we try to avoid redundancy in the representation of limits. Overall, the outline of this chapter is designed to gradually build on the previous steps rather than strictly separating the results which hold for the generic intruder model and the ones that do not.

4.1 Incorporation Check

Let us combine the two results from Sections 3.4 and 3.5 to check inductivity of a limit: given a limit L, we have to check whether $[\![L]\!] \supseteq \widehat{\text{post}}(L)$. So first, we have to compute the symbolic post for the limit and then check whether it is included in L. In contrast to Section 3.4 in which we considered arbitrary two limits for the inclusion check, we want to exploit the connection between the two limits in our use case. Recall that

$$\widehat{\text{post}}^s_\Delta(L) := \left\{ \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q' \parallel R\big) \;\middle|\; \nu\vec{x}.\big(\langle\Gamma\rangle \parallel Q\big) \to_\Delta \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q'\big) \in \mathbb{S}_s \right\}$$

for $b = \beta(\Delta) \cdot \gamma(\mathbb{I})^{s-1} + 1$ and $\text{sf}(L \otimes b) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$. Let $\widehat{\text{post}}^s_\Delta(L) = \{L_1, \cdots, L_n\}$. For the same reason that we merely had to prove decidability for the inclusion of two limits (rather than two downward-closed sets) as explained in the beginning of Section 3.4, it suffices to check that $L_j \subseteq L$ for every $1 \leq j \leq n$. For each L_j, we know that it is the result of taking one transition from $\nu\vec{x}.\big(\langle\Gamma\rangle \parallel Q\big)$.

Approximation Intuitively, the change resulting from one transition is rather local. Except for the process call to be reduced, the context stays the same and we simply add the continuation. So instead of checking the inclusion for both full limits, we only compare the differences.

We choose one process call for which we consider some transition leading to L_j for some j: $Q = Q' \parallel \mathbf{in}(\vec{z} : M).D[\vec{z}]$ where $D[\vec{z}]$ is an abbreviation for the continuation with which we can easily substitute for the right messages. By construction, $L \otimes b = \mathcal{C}[\mathsf{Q}[\vec{M}]]$ for some context $\mathcal{C}$. After pattern matching with a message $M[\vec{N}/\vec{z}]$ which might contain names from $\vec{c}$, the limit looks as follows:

$$L_j = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q' \parallel D[\vec{N}] \parallel R) = \mathcal{C}[D[\vec{z}]]$$

where the names $\vec{c}$ are part of $D[\vec{N}]$. So we have to check whether

$$[\![\nu\vec{x}.(\langle\Gamma\rangle \parallel Q' \parallel D[\vec{N}] \parallel R)]\!] \subseteq [\![\nu\vec{x}.(\langle\Gamma\rangle \parallel Q' \parallel \mathbf{in}(\vec{z} : M).D[\vec{x}] \parallel R)]\!]$$

of which a lot of parts are actually the same. If we exploit this observation, we can reduce the check to merely considering the changed part, which is $P := \langle\Gamma\rangle \parallel D[\vec{N}]$, is included in

$$E := \langle\Gamma\rangle \parallel \mathbf{in}(\vec{x} : M).D[\vec{z}] \parallel R.$$

E contains Γ as knowledge is persistent, $\mathbf{in}(\vec{z} : M).D[\vec{z}]$ since it is not needed to cover Q' and R as they are all decorated by ω.
Actually, this amounts to checking membership of a process in a limit which we can again check with Theorem 3.4 without recursion. From Lemma 9, we know that there is an n such that $P \sqsubseteq_{\mathsf{kn}} \lceil E\rceil^n$. This approximation to test inclusion is called *incorporation check*.

Example 11 (Incorporation Check)**.** Let us consider the limit L for Example 2 given in Section 3.6. We want to check inductivity for the first step, i.e. that the continuations of $\mathsf{A}_1[\text{-}]$ are captured by the invariant. As we consider the process call $A_1[\text{-}]$ which occurs in L and its continuations in L_1, we will handle the other limits symbolically by simply keeping L_2 as a sublimit:

$$L = \nu a, b, k_{as}, k_{bs}.\big(\langle(a, b)\rangle \parallel \mathsf{A}_1[a, b, k_{as}] \parallel \mathsf{B}_1[a, b, k_{bs}] \parallel \mathsf{S}[a, b, k_{as}, k_{bs}]^\omega \parallel \\ (\nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2^\omega)^\omega\big).$$

We know that

$$\mathsf{A}_1[\text{-}] \to \nu n_a.(\langle(n_a, b)\rangle \parallel A_2[a, b, k_{as}, n_a] \parallel A_1[a, b, k_{as}]) =: P.$$

We will see in Section 4.3 that extending the limit by 1 suffices to check inclusion as $A_1[\text{-}]$ does not take any input and all possible transitions of (this) $A_1[\text{-}]$ are checked.

$$\begin{aligned} L \otimes 1 = \nu a, b, k_{as}, k_{bs}.\big(&\langle(a, b)\rangle \parallel \mathsf{A}_1[a, b, k_{as}] \parallel \mathsf{B}_1[a, b, k_{bs}] \parallel \\ &\mathsf{S}[a, b, k_{as}, k_{bs}] \parallel \mathsf{S}[a, b, k_{as}, k_{bs}]^\omega \parallel \\ &(\nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2 \parallel L_2^\omega) \parallel \\ &(\nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2^\omega)^\omega\big) \end{aligned}$$

Hence, $\widehat{\mathrm{post}}(L)$ will contain the limit $L' = \mathcal{C}[P]$ where $\mathcal{C}[\bullet]$ is the same as $L \otimes 1$ but with a hole at the place where $\mathsf{A}_1[\text{-}]$ is. We want to prove inclusion between L and $L' = \mathcal{C}[P]$. The denotational semantics of L is the same as the one of $L \otimes 1 = \mathcal{C}[\mathsf{A}_1[\text{-}]]$. Hence, we try to keep the context in both limits and check whether P can be matched using the iterated parts, the process call in which the transition originated and the knowledge:

$$\begin{gathered}\nu n_a.(\langle(n_a, b)\rangle \parallel A_2[a,b,k_{as},n_a] \parallel A_1[a,b,k_{as}]) \\ \sqsubseteq_{\mathsf{kn}} \\ \langle(a,b)\rangle \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{S}[a,b,k_{as},k_{bs}]^\omega \parallel \langle n'_a\rangle \parallel (L_2[n'_a/n_a]^\omega) \parallel \\ (\nu n_a.(\langle n_a\rangle \parallel \mathsf{A}_2[a,b,k_{as},n_a] \parallel L_2^\omega)^\omega\end{gathered}$$

where n'_a indicates a renamed message (originally n_a) to avoid name clashes. It is straightforward to see that the knowledge embedding holds: A_1[-] can be matched immediately while b is covered by the pair (a, b). The remainder $\nu n_a.(\langle n_a\rangle \parallel A_2[a,b,k_{as},n_a])$ can be covered by the iterated limit in the last line. So we see that there is no need to rematch the whole context in this example but the incorporation check works.

This incorporation check suffices in many cases but there are corner cases for which it does not work, i.e. the check returns false even though the limit is included.

Example 12 (Parallel replaces Union)**.** Consider two limits L_1 and L_2 that represent an inductive invariant. Assume that there is some process $P_1 \in [\![L_1]\!]$ such that $P_1 \to P_2 \in [\![L_2]\!] \setminus [\![L_1]\!]$. Obviously, the incorporation check for the same limit L_1 fails and it is not applicable for L_2 as no common context is available. This means that the incorporation check is a sound but incomplete inclusion check.

There are two obvious ways to side-step this problem. We currently do not consider unions of ideals but rather parallel compositions: $[\![L_1]\!] \cup [\![L_2]\!] \subseteq [\![L_1 \parallel L_2]\!]$ by downward-closure of $[\![\text{-}]\!]$. Equipped with this over-approximation, we can try to find a single limit as inductive invariant for which the incorporation check is applicable. This approach is incomplete as there are protocols whose reachable state space can only be captured by inductive invariants consisting of several ideals. Despite of the incompleteness, we were able to find inductive invariants consisting of single limits for all the benchmarks we have considered. We can imagine a second possibility with the same effect: adapting the syntax and semantics of limits to incorporate a choice-construct, i.e. $+$, would render the use of a common context possible and enable an incorporation check. As the current approach suffices for all benchmarks considered, we leave the design and implementation of the latter idea for future work. Even with the latter theory, there are pathological examples for which the incorporation will still fail.

Example 13 (Swapping parameters)**.** Consider the following single process call definition which simply swaps its parameters: $\mathsf{Q}[x,y] := \tau.\mathsf{Q}[y,x]$ and the initial configuration: $P = \nu a, b.\mathsf{Q}[a,b]$. Note that P itself is an invariant for P:

$$\nu a, b.\mathsf{Q}[a,b] \quad \to \quad \nu a, b.\mathsf{Q}[b,a] \equiv P$$

by renaming a to b and vice versa. But the incorporation check fails as both names belong to the context: $\mathsf{Q}[b,a] \not\sqsubseteq_{\mathsf{kn}} \mathsf{Q}[a,b]$.

Overall, the incorporation check might not be complete but serves its purpose by decreasing the computational workload in most cases.

4.2 Irreducible Knowledge

In this section, we elaborate on how to check whether $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ for two sets of messages Γ_1 and Γ_2. We assume the symmetric intruder model $\mathbb{I}_{\mathsf{sy}}$. However, we will hint at possible generalisations in the sense of additional requirements for a general intruder model to apply the following results. We need the following ingredients for the $\leq_{\mathsf{kn}}$-check:

- a terminating and confluent rewriting system for sets of messages which induces unique normal forms, denoted by $\mathrm{ird}(\Gamma)$
- a way to reduce the check whether $\Gamma_1 \vdash M \implies \Gamma_2 \vdash M$ for every message M to a finite set of messages, as done with Proposition 1
- an algorithm to generate normal forms from arbitrary sets of messages that is based on the destruction rules of $\vdash$
- a new derivability relation $\vDash$ consisting of the construction rules of $\vdash$ only and a soundness result:
$$\forall \Gamma, M\colon\ \mathrm{ird}(\Gamma) \vdash M \iff \mathrm{ird}(\Gamma) \vDash M.$$

Note that Proposition 1 already gives one ingredient for the general intruder model. We will present how to instantiate the remaining ingredients for the intruder model for symmetric encryption $\mathbb{I}_{\mathsf{sy}}$.

4.2.1 Rewriting System for $\mathbb{I}_{\mathsf{sy}}$

In [DOT17], a confluent and terminating rewriting system for $\mathbb{I}_{\mathsf{sy}}$ has been introduced. We will exploit the normal forms induced by this system to establish the algorithmics to compare two knowledge bases. Let us recall some definitions and basic properties from [DOT17].

Definition 33 (Rewriting System from [DOT17])**.** We define a rewriting system $\longrightarrow$ for sets of messages with the following inference rules.

$$\frac{}{\Gamma, (M, N) \longrightarrow \Gamma, M, N}\ R_P \qquad\qquad \frac{\Gamma, \mathsf{e}(M)_K \vdash K}{\Gamma, \mathsf{e}(M)_K \longrightarrow \Gamma, M, K}\ R_E$$

Lemma 24 (from [DOT17])**.** The rewriting system in Definition 33 is terminating and confluent.

Lemma 25 (from [DOT17])**.** Let Γ_1, Γ_2 be two sets of messages. If $\Gamma_1 \longrightarrow \Gamma_2$, then $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$.

Definition 34 (Irreducible Set)**.** A set of messages Γ is called to be *irreducible* if there is no Γ' such that $\Gamma \longrightarrow \Gamma'$.

By Lemma 24, we know that there is a unique and irreducible set of messages Γ' for every Γ such that $\Gamma \longrightarrow^* \Gamma'$. We denote Γ' by $\mathrm{ird}(\Gamma)$. With Lemma 25, every set of messages is knowledge equivalent to its irreducible set: $\Gamma \sim_{\mathsf{kn}} \mathrm{ird}(\Gamma)$.

Types of Rules We distinguish two different kind of rules in the derivability relation $\vdash$ for the symmetric intruder model: we call Rules P_R and E_R *construction* rules while Rules P_L and E_L are called *destruction* rules. Construction rules may also be called introduction rules as they introduce operators while destruction rules may also be called elimination rules as they eliminate operators. The terminology stems from the two different perspectives: construction and destruction deal with building messages from smaller messages and destructing them again while introduction and elimination deal with the use of operators. It is straightforward to see that the rewriting system with its rules Rules R_E and R_P imitates the destruction rules. Therefore, we will basically split $\vdash$ and use the destruction rules to compute irreducible sets while we use the construction rules to check whether a message is derivable from an irreducible set of messages. As we will use the latter construction that yields a new derivability relation, we first assume to have an irreducible set of messages for which we want to check derivability for some message.

4.2.2 Exploiting Irreducibility

The question to answer is whether, given two knowledge bases Γ_1 and Γ_2, $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ holds. By Lemma 25 and Proposition 1, it suffices to check whether $\mathrm{ird}(\Gamma_2) \vdash M$ for every message $M \in \mathrm{ird}(\Gamma_1)$. As we consider irreducible sets of messages, the natural question arises in which way we can restrict the rules of $\vdash$ without sacrificing possible derivations. Intuitively, we only want to consider construction rules. To this end, let us first consider the different shapes of messages in irreducible sets of messages.

Lemma 26. Let Γ be an irreducible set of messages and let $N \in \Gamma$. Then, $N = a$ for some basic name a or $N = \mathsf{e}(M)_K$ for some messages M and K and $\Gamma \not\vdash K$.

Proof. Towards a contradiction. Assume $N \in \Gamma$ does not have this form. We do a case analysis on the types of messages.

$N = (M_1, M_2)$**:** If Γ contains a pair, it is not irreducible which is a contradiction.

$N = \mathsf{e}(M)_K$ **and** $\Gamma \vdash K$**:** In this case, we can apply R_P and hence Γ is not irreducible which is a contradiction.

□

Let us consider a knowledge base Γ and its irreducible set $\mathrm{ird}(\Gamma)$. We know that the set of derivable messages is exactly the same: $\forall M : \Gamma \vdash M$ iff $\mathrm{ird}(\Gamma) \vdash M$. In order to have a reduced version of $\vdash$, we investigate which rule applications can occur in cut-free proof tree for $\mathrm{ird}(\Gamma) \vdash M$.

Lemma 27. Let Γ be an irreducible set of messages and M a message such that $\Gamma \vdash M$. Then, every cut-free derivation of M only contains applications of Rule $\mathrm{I}\textsc{d}$, P_R and E_R.

Proof. Obviously, Rule $\mathrm{I}\textsc{d}$ occurs at every leaf. Both introduction rules P_R and E_R can occur in the derivation tree as they simply combine messages.

Claim: neither Rule P_L nor E_L can appear. To prove this claim, we exploit the shape of messages proven in Lemma 26. Γ does not contain any pair by Lemma 26 and hence Rule P_L cannot occur in the proof tree. Γ only contains encrypted messages for which the key cannot be derived. But this is the premise of Rule E_L which is the reason why it cannot be applied in the derivation either. There are no more rules to consider, so the claim follows. □

This lemma enables us to establish a subrelation of $\vdash$, denoted by $\vDash$, for which derivability is the same for irreducible sets of messages.

Definition 35. We define a new derivability relation $\vDash$ to be the relation only consisting of the rules Rule Id, P_R and E_R from $\vdash$.

Corollary 6. Let Γ be an irreducible set of messages and M some message. Then, $\Gamma \vdash M$ iff $\Gamma \vDash M$.

Proof. Follows directly from Lemma 27. □

Algorithm 1 (Checking $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$.)**.** Assume that both knowledge bases Γ_1 and Γ_2 are given as irreducible sets. Then, we check whether every message N from $\text{ird}(\Gamma_1)$ can be derived from Γ_2: $\Gamma_2 \vDash N$. This new relation makes this check very easy: First, we check whether $N \in \Gamma_2$. If so, we are done. If not and it is a basic name, the claim does not hold. If not and we have an encrypted message $\mathsf{e}(M)_K$, we split it and check for M and K individually. Possibly, M or K is a pair which we split and we proceed in the same way.

4.2.3 Computing Irreducible Sets

The algorithm basically is a fixedpoint iteration imitating the rewrite rules of $\longrightarrow$.

Algorithm 2 (red(-))**.** Let Γ be a set of messages.
Apply the following steps until nothing changes:

1. Take all pairs out of Γ, split them and put the components back into Γ.
2. For every encrypted message $\mathsf{e}(M)_K \in \Gamma$, check whether $\Gamma \vDash K$. If so, take $\mathsf{e}(M)_K$ out of Γ and put M and K into Γ. If not, $\mathsf{e}(M)_K$ stays in Γ.

Lemma 28 (Correctness of Algorithm 2)**.** For any knowledge base Γ, $\text{red}(\Gamma) \sim_{\mathsf{kn}} \Gamma$ and $\text{red}(\Gamma)$ is irreducible.

Proof. First, we prove termination. By construction, Algorithm 2 imitates rewriting steps of $\longrightarrow$. Therefore, the fixedpoint iteration eventually stabilizes as the rewrite system is terminating as $\vDash$ is a subrelation of $\vdash$. It remains to prove that the result is irreducible. There are no pairs as they will be split by definition. One could imagine that two encrypted messages could block each other's rewriting. The point of interest is the use of $\vDash$ instead of $\vdash$ in the algorithm. It is easy to see that it would be sound if we used $\vdash$. So the question arises whether the use of $\vDash$ will prevent us from rewriting at any point. Therefore consider some derivation $\Gamma, \mathsf{e}(M)_K \vdash K$ to obtain the key for an encrypted message. Again, we check which rules could occur in the derivation tree. As we split all pairs, P_L cannot occur. All the others may, especially E_L. The derivation is as follows.

$$\frac{\Gamma, \mathsf{e}(M)_K, \mathsf{e}(M')_{K'} \vdash K' \qquad \Gamma', \mathsf{e}(M)_K, \mathsf{e}(M')_{K'}, M', K' \vdash K}{\Gamma', \mathsf{e}(M)_K, \mathsf{e}(M')_{K'} \vdash K}\ \text{E}_L$$

The pattern to consider here is the left premise of this rule application. Essentially, we again want to derive a key for an encrypted message. We know that every derivation is finite so even for a sequence of E_L applications, there is at least one in the derivation subproof, for which no application of E_L is needed, with which the whole sequence starts. For this special subproof, there are only occurrences of Rule Id, P_R and E_R and we can therefore also derive this first key by using $\vDash$. By this, we can derive the second key in the next iteration of the algorithm. Proceeding inductively leads to the irreducible set of messages even though we used $\vDash$ to derive, respectively compose, the keys.

Second, we show that all obtained intermediate knowledge bases are knowledge equivalent to Γ. Let $\Gamma = \Gamma_1, \cdots, \Gamma_n = \mathrm{red}(\Gamma)$ be the intermediate knowledge bases obtained when applying the algorithm. The algorithm imitates the rewriting steps of $\longrightarrow$ and hence $\Gamma \sim_{\mathsf{kn}} \Gamma_j$ for every $0 < j \leq n$ by Lemma 25. Hence $\mathrm{red}(\Gamma) \sim_{\mathsf{kn}} \Gamma$. □

Lemma 29 (Checking $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ is polynomial)**.** For two arbitrary knowledge bases Γ_1 and Γ_2, we can check whether $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ in polynomial time.

Proof. Recall that our approach to check $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ consists of two steps. First, we reduce both knowledge bases to be irreducible by Algorithm 2 which we indicate by a primed version of the bases. Second, check whether $\Gamma'_2 \vDash M$ for every $M \in \Gamma'_1$. It is straightforward that the Algorithm 1 comprises the last check in polynomial time with respect to the size of M and Γ'_2. It remains to argue that Algorithm 2 produces the irreducible sets in polynomial time and that the size of the irreducible sets is also polynomial. Considering the size, we only split messages and never introduce new symbols, hence the result is of linear size. The number of iterations is at most the number of derivation steps to obtain every single message in the irreducible result. The length of derivations is polynomial for $\mathbb{I}_{\mathsf{sy}}$. Every iteration iterates once through the current set which is of polynomial size. Overall, Algorithm 2 needs polynomial time. □

Irreducible Knowledge is Irreducible We might want to prune further: consider the irreducible knowledge base $\Gamma = \langle a \rangle \parallel \langle \mathsf{e}(a)_c \rangle$. Even though we cannot reduce the second message, we already know the encrypted content a. As we will consider knowledge bases evolving over time, we have to ask whether it can make any difference to have the second message in terms of derivability. Intuitively, it does not. There are two cases to consider: First, if we never obtain the key c, the message will not be of any use. Second, suppose we will obtain c at some point. However, it could be important for a knowledge embedding to break:

$$\nu a.\nu c.\langle a \rangle \parallel \langle \mathsf{e}(a)_c \rangle \not\sqsubseteq_{\mathsf{kn}} \nu a.\langle a \rangle \text{ but } \nu a.\langle a \rangle \sqsubseteq_{\mathsf{kn}} \nu a.\langle a \rangle$$

as νc is to be omited in order to satisfy the constraint that every name is used at least once in its scope.

Second, $\mathsf{e}(a)_c$ adds a component for composing and therefore deriving new messages:

$$a, \mathsf{e}(a)_c \vDash \mathsf{e}(a)_{\mathsf{e}(a)_c} \text{ but } a \nvDash \mathsf{e}(a)_{\mathsf{e}(a)_c}.$$

4.3 Pattern Matching

In this section, we will describe how to efficiently implement the pattern matching which is needed for $\widehat{\mathrm{post}}(\text{-})$. To this end, we will refine the message size constraint s and introduce a sizing function for patterns.

The Setting In Section 3.5, we have shown that computing the symbolic version of $\mathrm{post}(L)$ for some limit L, i.e. $\widehat{\mathrm{post}}(L)$, can be reduced to considering the fixed part of $L \otimes b$ only. Hence, computing $\widehat{\mathrm{post}}(L)$ can be diminished to the following problem: given $P = \nu\vec{x}.(\Gamma \parallel Q)$ where Γ is irreducible and Q is a set of processes, we want to compute $\mathrm{post}(P)$.

Let us outline the steps to consider every possible successor:

- we consider every distinct process call $\mathsf{Q}[\vec{y}] \in Q$ with its corresponding definition
- for each process call, we find a number of actions with input prefixes and continuations
- for each action, we have to produce every possible message eligible for the pattern matching in these prefixes.

This number is finite because of a finite number of ground messages to consider and the size bound s we imposed on messages. We assume all patterns to be implementable for this section. For example, this can be checked as described in [DOT17] for the symmetric intruder model $\mathbb{I}_{\mathsf{sy}}$.

4.3.1 Patterns and Variables

We formally introduced input prefixes $\mathbf{in}(\vec{z} : M)$ with a vector of names to be bound and a pattern M already. Recall that we used the syntax for messages. There is no need to distinguish messages and patterns in the π-calculus, but it is more convenient to do so for terminology. Hence, we also define variables for pattern matching as the names that are bound when applying a pattern match. One can think of them as a subtype of names in the π-calculus.

Definition 36 (Variables)**.** Let $\mathbf{in}(\vec{z} : M)$ be an input prefix. We call $\vec{z}$ the *variables* of the pattern M and denote them by $\mathrm{fv}(M)$. Names in M, that are not variables, are called *constants*.

Note that we can only refer to $\mathrm{fv}(M)$ in the context of an input prefix.

Intuitively, it is clear when a message N can be matched to a pattern M. Basically, we want the syntax tree of M to be a rooted subtree of N so that the variables in M are bound to the corresponding subtrees of N.

Definition 37 (Pattern Match)**.** A message N can be matched to a pattern M with variables $\vec{z}$ if there is a list of messages $\vec{L}$ such that $N = M[\vec{L}/\vec{z}]$. This is denoted by $N \succ M : \theta$ where θ indicates the needed substitutions. A message M is called *eligible* for the pattern M. We may omit θ if not needed.

Let $\mathbf{in}(\vec{z} : M).P$ be an action. We see that $\vec{L}$ are the messages to substitute for $\vec{z}$ in P when matching N, leading to $P[\vec{L}/\vec{z}]$. This definition also ensures that constants in the pattern are matched instantly by N: if there is a position in the pattern M where the basic name a occurs, it ensures that the same a is in the same spot in N.

4.3.2 Forwarding

Prior to the actual pattern matching mechanism, we want to motivate a preprocessing step. This is a frequent pattern in protocols: some principal receives a message and sends it without modifications. It is easy to see that such a step does not contribute to the reduction semantics in our calculus. Hence, we can omit this kind of forwarding without sacrificing soundness. We want to rule out this kind of pattern which can be generalised to some extent. Note that this is a static analysis step and we can hence not consider some possible knowledge base from the environment but only consider a single input prefix and continuation.

Definition 38 (Simple Forwarding for $\mathbb{I}_{\mathsf{sy}}$)**.** Consider the following action as it may occur in a process call definition:

$$\mathbf{in}(\vec{x} : M).(\nu\vec{z}.(\langle\Gamma\rangle \parallel Q))$$

where Γ is irreducible. We say that some $x \in \vec{x}$ is *simply forwarded* if $M \vdash x$, $x \in \Gamma$ and $x \notin \mathrm{fv}(\Gamma \setminus \{x\})$.

Remark 5 (Simple Forwarding and Encryption Oracle)**.** Note that $x \in \Gamma$ requires x to appear as a singleton on top level and $x \notin \mathrm{fv}(\Gamma \setminus \{x\})$ ensures that x is not used at any other place. In case the first two conditions hold and the third does not, this action is an encryption oracle.

If some variable $x \in \vec{x}$ is simply forwarded in some action, we simply omit x since it does not change the reduction semantics in our calculus. To be very precise, there is a slight change. The intruder might introduce messages leading to $(\nu c.\langle c\rangle)^{\omega}$ in the invariant but this does not have impact on subsequent transitions as the intruder can introduce any desired number of nonces in every step. Formally, we could easily remedy to this obstacle by amending the reduction semantics in Definition 5 by requiring the name restrictions $\vec{z}$ in the successor only to contain names that have been used in the continuation of the action.

4.3.3 All Messages up to s

We first consider a naive approach of computing all possible messages. We exemplify the approach for the symmetric intruder model $\mathbb{I}_{\mathsf{sy}}$ and show that it is infeasible even for small message size bounds s.

Algorithm 3 (Messages up to s)**.** Consider an irreducible knowledge base Γ. The notation $\Gamma \vdash_t \Gamma'$ is used to denote that $\Gamma \vdash \Gamma'$ and the size of all messages in Γ' is t. We use a parametrised notation $\Gamma \vdash_t \Gamma_t$ for this algorithm. $\Gamma(t)$ contains all messages of size t in Γ: $\Gamma(t) = \{M \in \Gamma \mid \mathrm{size}(M) = t\}$. Solely names have size 1 and therefore: $\Gamma_1 = \Gamma(1)$. From this starting point, we construct derivable messages of larger size recursively:

$$\Gamma_{t+1} = \bigcup_{\substack{M_1 \in \Gamma_t \\ M_2 \in \Gamma_{t'} \text{ for } t' \leq t}} \{\mathsf{e}(M_1)_{M_2}, \mathsf{e}(M_2)_{M_1}, (M_1, M_2), (M_2, M_1)\} \cup \Gamma(t+1)$$

As we considered an irreducible set of messages, we know that $\vDash$ and $\vdash$ coincide. The rules of $\vDash$ only allow us to derive messages by combining messages to pairs or encryptions. This is the reason why we can construct every message up to size s which is derivable from Γ with this algorithm.

Example 14. Given the irreducible knowledge base $\Gamma = a, b, \mathsf{e}(c)_k$, let us compute messages up to size 3 (and hence omit the set notation).

$$\Gamma_1 = a, b \qquad \Gamma_2 = (a,a), \mathsf{e}(a)_a, (a,b), \mathsf{e}(a)_b, (b,a), \mathsf{e}(b)_a, \mathsf{e}(c)_k$$

$$\begin{aligned}\Gamma_3 =\,&((a,a),a), ((a,a),b), ((a,a),(a,a)), ((a,a),\mathsf{e}(a)_a),\\ &((a,a),(a,b)), ((a,a),\mathsf{e}(a)_b), ((a,a),(b,a)),\\ &((a,a),\mathsf{e}(b)_a), ((a,a),\mathsf{e}(c)_k), ((a,a),a), ((a,a),b),\\ &((a,a),(a,a)), ((a,a),\mathsf{e}(a)_a),\\ &((a,a),(a,b)), ((a,a),\mathsf{e}(a)_b), ((a,a),(b,a)),\\ &((a,a),\mathsf{e}(b)_a), ((a,a),\mathsf{e}(c)_k), \cdots\end{aligned}$$

Intentionally, we did not state *all* messages up to size 3 as they are numerous. Until now, we merely considered the first message of Γ_2 as the first component of a pair with all possibilities.

So let us check how many of these messages are useful for a specific input prefix.

Example 15. Given the irreducible knowledge base $\Gamma = a, b, \mathsf{e}(c)_k$ and the input pattern $\mathbf{in}(x, y : (x, \mathsf{e}(y)_k))$. Let us consider all messages up to size 3.
The subpattern $\mathsf{e}(y)_k$ can only be matched by $\mathsf{e}(c)_k$ and hence y will be bound to c for every valid pattern match. We can pair this encryption with any message of size up to 2 without exceeding the size bound. Therefore, x can be bound to any of the following messages: $a, b, (a, a), \mathsf{e}(a)_a, (a, b), \mathsf{e}(a)_b, (b, a), \mathsf{e}(b)_a, \mathsf{e}(c)_k$

This example demonstrates that all messages up to some size outnumber the eligible messages as they all need to satisfy the constraints given by the constants in a pattern. We therefore elaborate on ways of pruning the message space by investigating on the input pattern and especially its constants. In preparation for this, we introduce a concept which can be considered to be a more fine-grained version of the size bound s for messages originally suggested in [DOT17].

4.3.4 A Sizing Function for Variables

For decidability, a general size bound for messages is sufficient and as we did not consider the algorithmics, it was a neat way not to deal with unnecessary case distinctions which would have obscured the result. For computability, a more fine-grained version of a size bound is advantageous as it enables us to be very specific about the size of messages to be bound in pattern matches. Therefore, we make use of an additional sizing function which indicates the maximal size of a message bound to a variable in a pattern.

Constructing these bounds resembles the process of computing a size bound for messages in general. By computing a bound, we mean a natural one in the sense that honest executions of the protocol are possible. Natural (general) size bounds will always be at least 2. For 0, we cannot have any messages whereas there is no way to encrypt messages for a size of 1. For the general size bound, one considers such an honest run and annotates which variables are considered to be nonces, i.e. of size 1. Then, one propagates this information due to the execution of the protocol and hence obtains a maximum size for messages. Keeping the intermediate information about the sizes enables us to construct a sizing function from it. This function can also be used to enforce the size bound on messages in input prefixes. It is straightforward to amend the sizing function in a way such that the size bound for messages becomes obsolete. We simply have to compute the size and decrease the size annotations if the size bound could be exceeded by the message we use for the pattern match.

4.3.4.1 Pattern Matching with Sizing Functions

We first present the high-level idea and then establish the theory formally. In our decidability result, we established a large number to extend the limit with and then only considered the fixed part to compute $\widehat{\text{post}}$. We needed this large number as we considered all input patterns simultaneously and did not look into the input patterns individually. As stated in the definition of $\widehat{\text{post}}$, we extend with the same number for every input prefix but the latter is determined by the biggest input prefix in some sense. Hence, one improvement is to keep individual extension factors for every input prefix but there is more potential for improvement.

The following procedure is applied for every action of every process call in L. We then consider a specific action with its input prefix and its pattern and continuation. In order to investigate on the format of the pattern, we find out which constants of the pattern are derivable from the knowledge base. By this, we can see which parts of the pattern can or have to be constructed and which ones have to stem from the knowledge base. Based on these observations, the limit is extended appropriately and all possibilities for the pattern match are considered. For every resulting continuation, we check whether the latter combined with the fixed part is in L. In the following sections, we explain the different steps in depth.

Definition 39 (ζ, $\text{traces}^{\zeta}_{\Delta}(\text{-})$, (ζ, k)-bounded)**.** A *sizing function* maps (some) names to sizes:

$$\zeta : \mathcal{N} \rightharpoonup \mathbb{N}.$$

We require every substitution θ to respect ζ, i.e. for every name x, $\text{size}(\theta(x)) \leq \zeta(x)$. Note that pattern matching uses substitution and hence variables can also have sizes. By definition, restrictions can only have size 1. We extend the sizing function in a natural way:

$$\zeta(\mathsf{e}(M)_N) = \zeta((M, N)) = 1 + \max(\zeta(M), \zeta(N))$$

Let $\text{traces}^{\zeta}_{\Delta}(P)$ denote all traces respecting the sizing function, i.e. in every step all substitutions respect ζ. A process is (ζ, k)-bounded if $\text{traces}^{\zeta}_{\Delta}(P) \subseteq \mathbb{D}^{*}_{k}$.

Example 16. For an input prefix, we abuse notation and write

$$\mathbf{in}((m : \text{size}\,1) : (m, n))$$

to denote that $\zeta(m) = 1$.

4.3.4.2 The Pattern and its Constants

Example 17. Let us consider the following input prefix of $\mathsf{Q}[\vec{M}]$: $\mathbf{in}(x : \mathsf{e}(x)_k)$ and assume that $\mathsf{Q}[\vec{M}]$ is part of some limit L. Of course, the process call $\mathsf{Q}[\vec{M}]$ requires the key k. But for the intruder, there are two possibilities. If there is a knowledge base of any expansion of L in which k is revealed, we can construct a great number of messages which are eligible for the pattern match. But if not, we can only use exactly the messages with this pattern in the (irreducible) knowledge base which could be $\{\mathsf{e}(a)_k, \mathsf{e}(b)_k, c\}$ for instance. In this case, we only need to check for two different messages. Additionally, the intruder cannot generate messages using intruder names that are eligible for the pattern.

To decide which case of the latter example applies, we need to find a way to know whether such a key or any other component in a pattern match can be revealed or not. Surprisingly, the absorption axiom enables us to prove that expanding by 1 is enough to decide this. In case the key (or another part of the pattern) is derivable, we will apply the idea for the decidability result and expand as often as needed to have a distinct name with the same path on every position of the pattern. In order to keep the factor as small as possible, we will make use of the sizing function for variables.

Let us consider Example 17 again. We want to examine whether k is derivable in any instance of L. As k might occur in a process call in an iterated part of L, there is a number of k's. It does not matter which k is revealed as they are in some way corresponding to each other if we rename the expansion.

4.3.4.3 Corresponding Messages

We will establish a relation between messages to recognise corresponding messages. Thereby, we divide messages into classes and it suffices to analyse whether one member of this class is derivable. We want to find out for two structured messages whether they are corresponding to each other in the sense that the names used in the same position of both messages actually stem from the same name in the limit.

Example 18. Consider a limit $L = (\nu x, y.\cdots)^\omega$ where we merely consider the names. We expand L by 2 and obtain $\nu x_1, x_2, y_1, y_2.(\cdots \parallel \cdots)$ by keeping the original names as prefixes. Even though $\mathsf{e}(x_1)_{(y_1,y_2)}$ and $\mathsf{e}(x_2)_{(y_2,y_1)}$ are different, these messages correspond in some way as the names all stem from the same original name.
In contrast, $\mathsf{e}(x_1)_{y_1}$ and $\mathsf{e}(x_1)_{y_2}$ should not be considered corresponding as both names stem from the same component of the expansion in the first message but they do not in the second case.

Example 19. More precisely, we also need to consider nested expansions. Let $\nu x.(\nu y.\cdots)^\omega$ be a limit which leads to $\nu x, y_1, y_2.\cdots$ if one expands by 2. In this case, $\mathsf{e}(x)_{y_1}$ and $\mathsf{e}(x)_{y_2}$ should be considered to be corresponding.

Recall the source paths as defined in Definition 24. We introduce some terminology to distinguish ω's from other elements on the path. Let us explain how to map a path to a limit formally.

Definition 40 (Paths for Limits)**.** Let L be some limit and P_π be a source-path annotated instance of L. Let π' be some path from the annotations. With $\pi'(L)$, we denote the syntactical element that is found when applying π' to L. Since ω's expand to replications of components, a path might refer to a second component that does not exist in the limit. We therefore implicitly apply the path for the single ω-component.

Example 20. Consider the following limit

$$L = \nu x.(\nu y, z.(\mathsf{Q}[x, y] \parallel \mathsf{Q}[z, x]))^\omega$$

and the instance with its path annotations

$$\begin{aligned} \lceil L\rceil^2 = \nu x_\varepsilon.(&\nu y'_{00}, z'_{000}.(\mathsf{Q}[x_\varepsilon, y'_{00}]_{0000} \parallel \mathsf{Q}[z'_{000}, x_\varepsilon]_{0001}))_{00} \parallel_0 \\ &(\nu y''_{01}, z''_{010}.(\mathsf{Q}[x_\varepsilon, y''_{01}]_{0100} \parallel \mathsf{Q}[z''_{010}, x_\varepsilon]_{0101}))_{01}) \end{aligned}$$

and the different paths with their components:

$$\begin{aligned} \pi_1(L) &= x && \text{for } \pi_1 = \varepsilon \\ \pi_2(L) &= \omega && \text{for } \pi_2 = 0 \\ \pi_3(L) &= y && \text{for } \pi_3 = 00 \\ \pi_4(L) &= z && \text{for } \pi_4 = 010 \\ \pi_5(L) &= \mathsf{Q}[x, y] && \text{for } \pi_5 = 0000 \\ \pi_6(L) &= \mathsf{Q}[z, x] && \text{for } \pi_6 = 0001 \end{aligned}$$

Note that we implicitly apply the path operator to names in process calls.

Definition 41 (ω-paths and ω-labels)**.** Let L be some limit and P_π be some source-path annotated process that is an instance of L. A path π of P is an *ω-path* just if $\pi(L) = \omega$. Every component of a path π that refers to an ω is called *ω-label*.

Technically, ω-paths are sequences of numbers whereas ω-labels are numbers. We denote the ith element of some path π by $\pi[i]$ while $\pi[0..i]$ denotes the first $i+1$ elements. We introduced this terminology to have context-sensitive substitutions for ω-labels.

Definition 42 (Consistent ω-label-substitutions)**.** Let L be a limit and $(\pi_i)_{(0\leq i<n)}$ all paths from the path annotation for some process $P \in [\![L]\!]$. A substitution $\theta\colon (\pi_i)_{(0\leq i<n)} \to \mathbb{N}^*$ is called a *ω-label-substitution* just if only ω-labels in $(\pi_i)_{(0\leq i<n)}$ are substituted:

$$\theta(\pi_i) = \pi \text{ such that } \forall j\colon \pi_i[j] = \pi[j] \vee \pi_i[j] = \omega.$$

We call a ω-label-substitution θ *consistent* if there are no contradictions for common prefixes of paths:

$$\forall i, j < n, \text{ it holds that } \theta(\pi_i)[0..k] = \theta(\pi_j)[0..k] \text{ for every } k\colon \pi_i[0..k] = \pi_j[0..k].$$

Definition 43 (Shape of Messages)**.** Let M_1 and M_2 be two messages. We say that M_1 and M_2 have the *same shape* iff $\theta(M_1) = \theta(M_2)$ for $\theta\colon \mathrm{fn}(M_1) \uplus \mathrm{fn} \to \{a\}$ for some fresh name a.

Note that we cannot annotate whole messages but we can annotate the names used in messages as done for the process calls in Example 20.

Definition 44 (Corresponding Messages)**.** Let L be a limit, $n \in \mathbb{N}$ a number and $\nu\vec{x}.(\Gamma \parallel Q)$ be the standard form of $\lceil L \rceil^n$. Furthermore, let M_1 and M_2 be two messages of the same shape such that $\mathrm{names}(M_i) \subseteq \vec{x}$. We call M_1 and M_2 *corresponding* iff there is a consistent ω-label substitution θ for the paths of names in M_1 such that $\theta(\pi_1) = \theta(\pi_2)$ for every pair of paths for which there is a position in M_1 and M_2 where π_1 (π_2) is the path for the name in M_1 (M_2). We denote two corresponding messages by $M_1 \approx M_2$.

Note that even though the relation is not symmetric by definition, it is straightforward to construct a substitution θ' which renames ω-labels from M_2. It is also transitive and reflexive and hence an equivalence relation with which we can split messages into equivalence classes.

We used an existential quantification for simplicity in the definition. But it is straightforward to define an algorithm that constructs such a substitution θ if it exists and hence checks whether M_1 and M_2 are corresponding.

Algorithm 4 (Corresponding Messages)**.** For two messages M_1 and M_2 of the same shape, we iteratively construct a ω-label-substitution. Start with an empty substitution θ. For every position in M_1 (and M_2) that is filled with names n_1 and n_2 with the respective source paths π_1 and π_2.

- the lengths of π_1 and π_2 coincide.
- we add $\theta(\pi_1) = \pi_2$ to the substitution and check whether θ is still consistent. If so, we proceed. If not, M_1 and M_2 are not corresponding.

We are now turning back towards our goal, i.e. finding out how many distinct names are needed to generate every possible eligible message for a pattern without sacrificing completeness.

Definition 45 (Number of Distinct Instances)**.** Let M be a message constructed from path-annotated names. Let $(\pi_i)_{0\leq i<n}$ be the paths of all names occurring in M. We define a set

of sets of paths as indices so that there is common prefix up to some ω-label and all the indices for the latter are different:

$$\begin{aligned} S_M = \{ I \subseteq [0,n) \quad | \quad & \exists k: \\ & \big(\pi_i[k+1] = \omega \ \text{ for every } i \in I \quad \wedge \\ & (\pi'_i[0..k] = \pi''_i[0..k] \wedge \pi'_i[k+1] \neq \pi''_i[k+1]) \ \text{ for all } i', i'' \in I\big)\} \end{aligned}$$

Based on this set, the number of needed distinct expansions is the maximal cardinality of any of these sets.

$$\mu(M) = \max(|s| \mid s \in S_M).$$

Exploiting the absorption axiom, we will show that it suffices to expand $\mu(M)$ times in order to examine whether M (or any corresponding) message can be derived.

Lemma 30. Let L be a limit with $\mathrm{sf}(L) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$. With $\mathrm{sf}(\lceil L \rceil^n) = \nu\vec{x}_n.(\Gamma_n \parallel Q_n)$, we denote the standard form of an expansion by any $n \in \mathbb{N}$ for which we do not rename names in $\vec{x}$. Let M be a message containing at least one name from an iterated part, i.e. $\mathrm{fn}(M) \not\subseteq \vec{x}$. Let $m \in \mathbb{N}$ be a number such that $m \geq \mu(M)$, $m \geq 1$ and $\Gamma_{m+1} \vdash M$. Then $\Gamma_m \vdash M'$ for some message M' such that $M \approx M'$.

Proof. By Lemma 8, we know that $\nu\vec{x}_m.(\Gamma_m \parallel Q_m) \sqsubseteq_{\mathsf{kn}} \nu\vec{x}_{m+1}.(\Gamma_{m+1} \parallel Q_{m+1})$. Therefore and as they are stemming from the same limit, there is a renaming s.t.

$$\mathrm{sf}(\lceil L \rceil^{m+1}) \equiv \nu\vec{x}_m.\nu\vec{y}.(\Gamma_m \parallel \Gamma \parallel Q_m \parallel Q)$$

for some Γ and Q. Usually, we only know that $\Gamma_m \leq_{\mathsf{kn}} \Gamma_{m+1}$ but as no rewriting takes place, we can split the knowledge base in the part which is also present in the m-expansion and a remainder. This remainder stems from the increased expansion factor. As we require M to contain at least one name from an iterated part, we know that $m > 0$. As $m \geq \mu(M)$, we can assume that $\mathrm{names}(M) \subseteq \vec{x}_m$ by α-renaming.
We know that $\Gamma_m, \Gamma \vdash M$ and want to show that $\Gamma_m \vdash M$. Γ is the remainder of the larger expansion for which the set of names $\vec{y}$ is unique. As $m \geq 1$, we may split $\Gamma_m = \Gamma_{m-1} \parallel \Gamma'$ such that $\Gamma = \Gamma'[\vec{y}/\vec{y'}]$. Hence, we can apply Corollary 3 and know that $\Gamma_n \vdash M$ so the claim follows. ☐

With this lemma, we can reduce the factor to expand with to $\mu(M)$ to check whether some arbitrary message M is revealed. Let us consider our use case in which we want to check whether a part of a pattern in an input prefix can be constructed again. We see that all names used in this pattern stem from the same instance of expansion, i.e. there is a name with some path such that every other name's path is a prefix of the latter. Therefore, we know that $\mu(M) = 1$ for messages from input prefixes. This fact is stated in the following corollary.

Corollary 7. For any limit L and (sub-)message M in some input prefix, it is sufficient to expand L by 1 in order to decide whether M is derivable in any instance of L.

Remark 6 (On the Importance of Correspondence)**.** The concept of correspondent messages can easily be generalised to sets (or lists) of names. As we will see in the next section, we will encode the check for knowledge embedding as an instance for an SMT solver. There, the concept of corresponding names could be advantageous in combination with predicates in order to rule out several isomorphic solutions for mappings of names. This is one of the reasons why we presented all these concepts even though it finally can be reduced to expansions by 1.

By now, we know how to examine whether some part of the input prefix is derivable. Next, we want to explain how to use this systematically in input prefixes.

4.3.4.4 Computing the Set of Variables to be Filled

In any input prefix, there are two different types of variables. In case the surrounding structure of a variable cannot be derived from the knowledge base, we can only use eligible messages for this subpattern and such a variable will never be substituted for a freshly composed structured message. In case the surrounding structure can be derived, we may construct a message which will be substituted for this variable. Therefore, we have to generate a sufficiently high number of fresh names.

To figure out the needed expansion factor for every input prefix individually, we need to compute the set of variables of an input prefix which may be filled with fresh names. We compute this set in the following setting: We expanded a limit L by 1 and have an irreducible knowledge base Γ. By Corollary 7, Γ is enough to figure out whether parts of the input prefix will be revealed in any instance of L. The input prefix is denoted by $\mathbf{in}(\vec{x} : M)$.

Based on this, we can compute the set of variables to be filled. Intuitively, we need to generate fresh names for a variable if we can construct the remaining pattern from Γ. As a variable can occur several times in a pattern, we might come to different conclusions about its necessity for fresh names. In this case, the negative conclusion outweighs the positive one as the surrounding subpattern can not be constructed for at least one occurrence.

Example 21. As an example, one can consider the input prefix used by S in the encoding of the Otway-Rees protocol where m occurs three times, which is modelled in Section 5.2.

Algorithm 5. Given a pattern M with variables $\vec{x}$ and an irreducible knowledge base Γ, we describe an algorithm to compute the set of variables which can be filled with fresh names. We therefore annotate the syntax tree of the pattern. To do so, we use a set of symbols to indicate the options. Given a node C in the pattern tree, $\mathcal{C}$ indicates that a message is derivable or a pattern can be constructed from its components whereas $\mathcal{E}$ means that it can be matched with a message which is eligible. $\perp$ means that none of this is the case.
For simplicity, we traverse the tree twice but the whole process could be executed in one traversal. First, we prepare the base values of the annotation needed by the algorithm.

- For every (biggest) subtree T inducing a message N,
  $\mathrm{ann}(T) = \mathcal{C}$ if $\Gamma \models N$ and $\mathrm{ann}(T) = \perp$ if $\Gamma \not\models N$.
- For every variable x, set $\mathrm{ann}(x) = \mathcal{C}$.

Then, we propagate this information bottom-up. Every inner node in such a syntax tree has exactly two children. We describe the step of combining the information of two children C_1 and C_2 in an ancestor A for every function individually.

$$\mathrm{ann}(A) := \begin{cases} \mathcal{C} & \text{if } \mathrm{ann}(C_1) \neq \perp \ \wedge \ \mathrm{ann}(C_2) \neq \perp \\ \mathcal{E} & \text{if } \exists N \in \Gamma \text{ s.t. } N \succ A \ \wedge \ (\mathrm{ann}(C_1) = \perp \ \vee \ \mathrm{ann}(C_1) = \perp) \\ \perp & \text{otherwise} \end{cases}$$

This procedure results in an annotation of the syntax tree so that we know for every node whether the parts of the induced pattern can be constructed from its components or is only available through eligible message(s). In case the root is annotated by $\perp$, there is no way to construct a message which is eligible for the pattern. Otherwise, we compute two

sets of variables based on the annotations. The set A holds variables which can be filled whereas B holds variables which cannot since they are part of some subpattern for which only an eligible message exists.
We traverse the tree top-down and do the following in every inner node:
As long as it is annotated by $\mathcal{C}$, we keep on descending in both children. In case we find a node annotated by $\mathcal{E}$, we add all variables contained in the subpattern induced by this node to B. In case we reach a terminal node consisting of a variable, we add it to A. We do not need to consider $\perp$ as it can only occur for children of $\mathcal{E}$-annotated nodes where we do not pursue. The set of variables to be filled which we wanted to compute is exactly $V = A \setminus B$ because A contains all variables occurring in patterns we can construct and B consists of all variables which are fixed by eligible messages.

Correctness One interesting part is that we only consider messages in the irreducible knowledge base to check whether there is an eligible one. We would have to care about every derivable message which might be eligible. To rule out doubts about this, we state and prove the following lemma.

Lemma 31. *Let $M = \mathsf{e}(M_1)_{M_2}$ or $M = (M_1, M_2)$ be a pattern and Γ be an irreducible knowledge base. If there is a message N such that $N \succ M$ and $\Gamma \vdash N$ but $M \notin \Gamma$, the above algorithm would annotate this node with $\mathcal{C}$.*

Proof. As Γ is irreducible, we know that $\Gamma \vdash N$ and $\Gamma \vDash N$ are equivalent by Corollary 6. Therefore, we can split N in its components contained in Γ. Splitting the pattern in the same way leads to a set of subpatterns for which we have eligible messages (and variables which are denoted as $\mathcal{C}$). Thinking of an execution of the algorithm, it will either annotate a node with $\mathcal{C}$ in any case or find an eligible message in Γ and annotate it with $\mathcal{E}$. In both cases, the algorithm will propagate non-$\perp$ annotations up to M. As it only annotates with $\mathcal{C}$ and never with $\mathcal{E}$ in case none of the underlying results were $\perp$, we know that it is $\mathcal{C}$ as the annotations for M_1 and M_2 were non-$\perp$. This proves the claim. □

Possible Improvement In case a message is eligible for a subpattern, we could keep the witness θ so that we know the substitutions for this message. By this, we might be able to keep track of contradicting substitutions and rule them out quickly. Since this pattern does not occur very frequently in our benchmarks, we refrained from implementing it for this procedure. But we rule out contradicting substitutions immediately when computing the set of final substitutions. There are cases where variables are bound twice which is why we decided to keep track of variables that are bound by eligible messages so that we do not have to consider different substitutions. Indeed, we need to ensure that we can construct the appropriate messages for variables from B.
Nevertheless, one could not keep such substitutions in general. In case we needed to extend by more than 1, we assume that there are more eligible messages for different subpatterns in case it is annotated with $\mathcal{E}$. One could also think of tracking whether a pattern is constructable and eligible at the same time but we will also try to find eligible messages if a subpattern is considered to be constructable for convenience.

4.3.4.5 Messages to Fill Variables

Based on the variables that can be filled, we have to ensure that there are enough names from the same position in the original limit to construct these messages. We can use the sizing function ζ to determine the number of names that are needed to this end. The number of distinct names, which are needed, is then straightforward to compute as the sum of all names needed for every single variable:

$$t = \sum_{v \in V} 2^{\zeta(v)} - 1$$

Since we have to consider the case that every such name has the same path in the limit, this is also the factor we have to extend with because of the variables. Additionally, this is the number of different names the intruder can construct to influence the pattern match. Hence, we will incorporate a vector of t fresh names $\vec{c}$ according to the reduction semantics.

4.3.4.6 The Case of Multiple Eligible Subpatterns

There is one case left which might lead to increasing the factor to extend with.

Example 22. Consider the following process:

$$P = \nu k.(\nu m.\langle \mathsf{e}(m)_k \rangle)^\omega \parallel \mathbf{in}(x, y : (\mathsf{e}(x)_k, \mathsf{e}(y)_k)).\mathsf{Q}[x, y]$$

The two subpatterns $\mathsf{e}(x)_k$ and $\mathsf{e}(y)_k$ are equal up to renaming of variables. If we extended only once in this scenario, we would always have $\mathsf{Q}[m, m]$ as continuation where m stems from the iterated limit. Indeed, we should also consider the continuation $\mathsf{Q}[m_1, m_2]$ where m_1 and m_2 have the same path in the limit but stem from different expansions.

This example illustrates the issue we want to tackle. We refrain from formalising the relation these two subpatterns have as there are more subtle connections we have to incorporate. In a similar way as a message can be eligible for a pattern, a pattern can be eligible for another one. Therefore, we generalise this concept. Intuitively, we intend to find every two patterns M_1 and M_2 such that $M \succ M_1 \implies M \succ M_2$ for every message M. We formalise this in the following way.

Definition 46 (Eligible Patterns)**.** Let M_1 and M_2 be two patterns. We call M_1 eligible for M_2 if there is a substitution θ from variables of M_2 to messages and variables of M_1 such that $M_2\theta = M_1$.

This relation is reflexive, transitive and anti-symmetric up to renaming of variables. In Example 22, we considered a special case of this relation but we also want to give an example in which both subpatterns are not equal up to renaming of variables.

Example 23. Let x, y and z be variables and a and k be names. Consider the following two subpatterns:

$$\mathsf{e}(x, y)_k \text{ and } \mathsf{e}(z, a)_k$$

The second pattern is eligible for the first one as we can substitute the variable z for x and the name a for y.

This scenario considers only subpatterns for which we have eligible messages in the irreducible knowledge base. As we computed an over-approximation of variables we have to fill, we consider an over-approximation of eligible messages we have to compute. We can find the minimum extension factor by finding the longest chain of subpatterns that are eligible for each other. One could try to be a bit more fine-grained at this point and construct different versions for which the set of variables to be filled will be larger and the one of eligible messages we need is smaller for instance but the overhead to distinguish these cases will probably not pay off in the end.
Note that this feature is not implemented in our tool and such chains need not be input to obtain sound results.

4.3.4.7 Constructing Substitutions

By now, we computed the factor to extend with. When extending with some number greater than 1, there is more than one process call for every process call we considered in the limit. As all of them have the same behaviour in terms of possible one-step-transitions, it suffices to check only one of these. It remains to elaborate on how to compute the possible substitutions for the input prefix and the continuation. Even though we consider extension here, i.e. the version keeping ω's, we only consider the fixed part of the limit due to the definition of $\widehat{\text{post}}$ for the transitions. We will use the iterated parts to check whether the continuation is included in the limit.

We have an irreducible knowledge base Γ stemming from a sufficiently big expansion and an input prefix $\mathbf{in}(\vec{x} : M)$ with an annotated pattern M. In a naive approach, we would start at every terminal and construct all possible substitutions, combine them when going up and rule out contradicting substitutions. But we exploit to exploit the annotated properties of parts being constructable, derivable or eligible. We cannot construct messages for subpatterns that are marked with $\mathcal{E}$. Therefore, it suffices to compute all eligible messages for these subpatterns which are members of Γ by Lemma 31. Because of the extension, there might be more messages of this shape than before and hence we have to check this again but it will not overrule our result whether the subpattern is constructable, eligible or neither of both. All of them imply substitutions for the variables contained in this subpattern. There might be contradictions when combining substitutions. This is why we want to keep track of substitutions due to eligible messages and use this information when we construct messages for variables.

Algorithm 6 (Constructing Substitutions)**.** Traverse the syntax tree from top to bottom, doing the following — depending on the annotation.

$\mathcal{E}$: Compute all possible substitutions for variables contained in this subpattern by checking for eligible messages in Γ and stop descending.

$\mathcal{C}$: Keep on descending if possible. If not, it is a variable and we check for every substitution in the global state whether it can be derived, as it is needed to construct the message. If there are restrictions in the global state, we compute all possibilities respecting the size annotation.

Technically, we need to ensure that every node annotated with $\mathcal{E}$ is processed before the terminal nodes that are annotated with $\mathcal{C}$ so that all possible substitutions are in the global state. The shape of annotations allows this as $\mathcal{C}$ only occur higher in the syntax tree.

Eventually, inclusion needs to be checked for every single inclusion where we can use the incorporation check as described in Section 4.1.

4.4 Finding Candidates for Invariants

The applications we presented in Section 3.1 incorporated very expensive enumeration techniques which are a neat means to show decidability but are far from being applicable to practice. Recall that the idea was to enumerate all candidates for invariants and check their inductiveness and whether the initial protocol configuration is included.

Acceleration in the style of Karp and Miller There is a way to generate these candidates systematically by acceleration techniques that are similar to the Karp-Miller algorithm for Petri nets. [ZWH12] suggests a very general way of widening in the following way: given an initial protocol configuration, consider a transition sequence $P_1 \to^* P_2$ with $P_1 \sqsubseteq_{\mathsf{kn}} P_2$. The knowledge embedding entails the following two normal forms: $\mathrm{sf}(P_1) = \nu\vec{x}.(\Gamma \parallel Q)$ and $\mathrm{sf}(P_2) = \nu\vec{x},\vec{y}.(\Gamma \parallel Q \parallel P)$. As we can start the same transition sequence from P_2 again, we can produce $\nu\vec{y}.P$ arbitrarily often. Hence, we summarise by accelerating to $\nu\vec{x}.(\Gamma \parallel Q \parallel (\nu\vec{y}.P)^\omega)$.

This idea can be extended to limits. Suppose, we are given some initial limit L_0 that includes the initial configuration of a protocol. We explore the (finitely-branching) transition system that is induced by $\widehat{\mathrm{post}}$. For each of the limits that we reach, we check whether some limit on its path is included in this most recent one. If so, we apply a widening operator to obtain an over-approximation of the acceleration for this trace. As knowledge embedding is a wqo, we know that there cannot be infinite ascending chains and hence the procedure terminates. It returns a finite union of limits that is inductive by construction.

To get an intuition about how computationally intensive this procedure is, let us investigate about the number of inclusion checks in terms of the minimum and maximum branching factor as well as the length of traces in the induced transition system. Consider an initial configuration for which we know that the minimum (maximum) branching factor is b_1 (b_2) and the minimal (maximal) length of traces in the induced transition system is given by t_1 (t_2).

The number of inclusion checks is given by

$$C_j := \sum_{0 \leq i < t_j} i \cdot b_j^i = \frac{t_j \cdot b_j^{t_j+1} - t_j \cdot b_j^{t_j} + b_j}{(b_j - 1)^2}$$

where $j \in \{1, 2\}$. C_1 denotes the minimum number and C_2 the maximum number of inclusion checks. Of course, this is a rough approximation but gives the intuition we were aiming for. For inclusion checks, we already discussed the incorporation check that suffices in many cases. Nevertheless, the latter is not complete as discussed before. This observation led to the choice to replace union of two ideals by parallel composition if possible which we exploit in our coarser version of widening.

Coarser Widening To find candidates for invariants, we employ a form of widening as in abstract interpretation. For a given limit L, we want to construct some inductive invariant I such that $[\![L]\!] \subseteq [\![I]\!]$. The idea is quite straightforward: for every possible transition in L, we compute the corresponding limit given by the definition of $\widehat{\mathrm{post}}$ and check whether the limit is already included in L. If so, we are done. If not, we incorporate the continuation into L — possibly decorated with an ω if the same transition can happen again — and respect the common context thereby. We continue as long as there are no transitions left but may timeout before for practical reasons. Theoretically, it does terminate for the same reason of our domain being a wqo. In case there are no transitions left, we have found an inductive invariant.

Algorithm 7 (Coarse Widening — One Step)**.** Let L be a limit in standard form: $L = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$. With Q_R, we denote all process calls that are present in the syntax tree of R, i.e. not only the ones on the first level. We mark every process call in Q and Q_R with some label $k \in K$. For Q_R, we can thereby distinguish where the different process calls are stemming from.

1. Compute the factor to branch with, i.e. b and expand the limit L by b:

$$\mathrm{sf}(\lceil L \rceil^b) = \nu \vec{x}, \vec{y}.(\langle \Gamma' \rangle \parallel Q')$$

2. Let $\mathsf{Q}_k[\vec{M}]$ be any process call labelled by $k \in K$ such that

$$\mathrm{sf}(\lceil L \rceil^b) = \nu \vec{x}, \vec{y}.(\langle \Gamma' \rangle \parallel Q_k \parallel Q'')$$

   and define a context $\mathcal{C}[\text{-}]$ for the original limit:

$$\mathcal{C}[\mathsf{Q}_k[\vec{M}]] = \nu \vec{x}.(\langle \Gamma \rangle \parallel Q \parallel R).$$

   Now, collect all continuations of possible transitions for $\mathsf{Q}_k[\vec{M}]$ in some set D. Filter the continuations and only keep the ones which are not included in L already and obtain D'. We could annotate the different continuations with the transition that produced them to possibly backtrack concrete runs. In case D' is empty for every possible label $k \in K$, we are done.

3. We combine the continuations from D' to some process $P_{D'}$. Finally, we incorporate $P_{D'}$ into the limit L by adding it to the context C:

$$L' := \mathcal{C}[\mathsf{Q}_k[\vec{M}] \parallel P_{D'}^{\omega}]$$

   where we decorate $P_{D'}$ with an ω since $\mathsf{Q}_k[\vec{M}]$ can reproduce the continuations.

4. Check whether the security property holds for the amended limit L'.

   If so, proceed with the amended limit in the same way. If not, return and let the user check interactively how to proceed.

Note that there is no need to check the continuation of every process call contained in the expanded limit. We labelled the different process calls before expanding as another process call with the same label produces the same continuations up to α-renaming. Hence it suffices to only check one process call for each one occurring in the original limit.

The Case of Failure In case a limit violating the security protocol is returned, we have to interact and check why the property is violated: it is possible to backtrack how the different parts leading to the violation became part of the invariant based on the annotations. Note that we currently do not support this in the tool but it was always easy to see why the property fails in our experiments. There are two reasons for the algorithm to fail. First, the protocol does really violate the property and we have then found an actual run violating the property. Second, the inductive invariant only violates the property as it has been over-approximating too much and the property actually does hold. The latter can also occur due to the fact that we do not consider unions of ideals but single ideals as candidates for invariants.

Combining Continuations We have not explained yet how to "combine" the continuations in step 3 of Algorithm 7. Let us exemplify the approach with an actual protocol and explain heuristics to combine continuations in a smart way whilst generating the invariant.

Example 24 (Generating an Invariant)**.** In this example, we generate an invariant for a variant of the Yahalom protocol as presented in [DOT17]. As explained in Section 4.3, we will omit simple forwarding so that the protocol can be encoded as follows:

$$\begin{array}{lll}
\mathsf{S}_1[x,y,k_{xs},k_{ys}] & := & \mathbf{in}(n_x,n_y : (y,\mathsf{e}(x,n_x,n_y)_{k_{ys}})). \\
 & & (\nu k_{xy}.(\langle \mathsf{e}(y,k_{xy},n_x,n_y)_{k_{xs}}\rangle \parallel \langle \mathsf{e}(x,k_{xy})_{k_{ys}}\rangle \parallel \mathsf{S}_1[x,y,k_{xs},k_{ys}])); \\
\mathsf{A}_1[x,y,k_{xs}] & := & \tau.(\nu n_x.(\langle (x,n_x)\rangle \parallel \mathsf{A}_1[x,y,k_{xs}] \parallel \mathsf{A}_2[n_x,x,y,k_{xs}])); \\
\mathsf{B}_1[x,y,k_{ys}] & := & \mathbf{in}((n_x : \mathrm{size}(1)) : (x,n_x)). \\
 & & (\nu n_y.(\langle y\rangle \parallel \langle \mathsf{e}(x,n_x,n_y)_{k_{ys}}\rangle \parallel \mathsf{B}_1[x,y,k_{ys}] \parallel \mathsf{B}_2[n_y,x,y,k_{ys}])); \\
\mathsf{A}_2[n_x,x,y,k_{xs}] & := & \mathbf{in}(n_y,k_{xy} : \mathsf{e}(b,k_{xy},n_x,n_y)_{k_{xs}}).(\langle \mathsf{e}(n_y)_{k_{xy}}\rangle); \\
\mathsf{B}_2[n_y,x,y,k_{ys}] & := & \mathbf{in}(k_{xy} : \mathsf{e}(x,k_{xy})_{k_{ys}}).(\langle \mathsf{e}(n_y)_{k_{xy}}\rangle);
\end{array}$$

with the start configuration

$$\mathsf{B}_1[a,b,k_{bs}] \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{S}_1[a,b,k_{as},k_{bs}] \parallel \langle (a,b)\rangle.$$

where we make the names of the participants publicly available. This configuration is also the starting point to generate the invariant. We may omit process call parameters and write A_1[-] instead of $\mathsf{A}_1[a,b,k_{as}]$. Furthermore, note that x in most cases corresponds to a while y corresponds to b. First, we check for all input prefixes whether the pattern can be matched by any message. For S_1[-], this is not the case. For both A_1[-] and B_1[-], we can match the pattern. Now, we can choose the process call. The protocol design suggests that A takes the first step so we choose A_1[-]. With its τ-transition, there is only one continuation to consider:

$$\nu n_a.(\langle (a,n_a)\rangle \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{A}_2[n_a,a,b,k_{as}])$$

Since a and A_1[-] are already covered by the initial configuration, we only have to add

$$L_1 := \nu n_a.(\langle n_a\rangle \parallel \mathsf{A}_2[n_a,a,b,k_{as})$$

and decorate with an ω as A_1[-] may always reproduce it. We obtain:

$$\mathsf{B}_1[a,b,k_{bs}] \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{S}_1[a,b,k_{as},k_{bs}] \parallel \langle (a,b)\rangle \parallel L_1^{\omega}$$

Now, B_2[-] is the only one to fire transitions. The intended message for n_x is n_a which is a nonce. Hence we use the sizing function ζ to restrict n_x to be a nonce and do not use a global message size constraint as this requires n_x to be a nonce in $\mathsf{e}(x,n_x,n_y)_{k_{ys}}$. There are three possible messages which can be substituted for n_x: a, b and n_a. We combine these three continuations by only using one single fresh name and put the messages in parallel:

$$L_2 := \nu n_b.(\langle \mathsf{e}(a,n_a,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,b,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,a,n_b)_{k_{bs}}\rangle \parallel \mathsf{B}_2[n_b,a,b,k_{bs}]).$$

Since L_2 does use the name n_a from L_1, we add L_2 in L_1:

$$L_1 := \nu n_a.(\langle n_a\rangle \parallel \mathsf{A}_2[n_a,a,b,k_{as}) \parallel L_2^{\omega}.$$

Now, S_1[-] can use all these three messages produced by B_1[-] as input and we can combine the continuations:

$$L_3 := \nu k_{ab}.(\langle \mathsf{e}(b,k_{ab},b,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(b,k_{ab},a,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(b,k_{ab},n_a,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(a,k_{ab})_{k_{bs}}\rangle);$$

and augment L_2:

$$L_2 := \nu n_b.(\langle \mathsf{e}(a,n_a,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,b,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,a,n_b)_{k_{bs}}\rangle \parallel \mathsf{B}_2[n_b,a,b,k_{bs}] \parallel L_3^{\omega}).$$

For the next step, both $\mathsf{A}_2[\text{-}]$ and $\mathsf{B}_2[\text{-}]$ can match the pattern. We will again choose $\mathsf{A}_2[\text{-}]$ as suggested by the design of the protocol. The input prefix can only match one of the messages in L_3 which results in $\mathsf{e}(n_b)_{k_{ab}}$. The latter is actually subsumed by the continuations as produced by $\mathsf{B}_2[\text{-}]$:

$$L_4 := \langle \mathsf{e}(n_b)_{(n_a,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{(b,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{(a,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{k_{ab}}\rangle.$$

We add L_4 to L_3. There is no reason to decorate it with ω due to persistence of messages:

$$\begin{aligned} L_3 := \nu k_{ab}.(&\langle \mathsf{e}(b,k_{ab},b,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(b,k_{ab},a,n_b)_{k_{as}}\rangle \parallel \\ &\langle \mathsf{e}(b,k_{ab},n_a,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(a,k_{ab})_{k_{bs}}\rangle \parallel L_4). \end{aligned}$$

We could have lifted the messages not containing k_{ab} but we refrain from doing this. Overall, the generated invariant is:

$$\mathsf{B}_1[a,b,k_{bs}] \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{S}_1[a,b,k_{as},k_{bs}] \parallel (a,b) \parallel L_1^{\omega}$$

$$\begin{aligned} L_1 &:= \nu n_a.(\langle n_a\rangle \parallel \mathsf{A}_2[n_a,a,b,k_{as}) \parallel L_2^{\omega} \\ L_2 &:= \nu n_b.(\langle \mathsf{e}(a,n_a,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,b,n_b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(a,a,n_b)_{k_{bs}}\rangle \parallel \mathsf{B}_2[n_b,a,b,k_{bs}] \parallel L_3^{\omega}) \\ L_3 &:= \nu k_{ab}.(\langle \mathsf{e}(b,k_{ab},b,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(b,k_{ab},a,n_b)_{k_{as}}\rangle \parallel \\ &\qquad \langle \mathsf{e}(b,k_{ab},n_a,n_b)_{k_{as}}\rangle \parallel \langle \mathsf{e}(a,k_{ab})_{k_{bs}}\rangle \parallel L_4) \\ L_4 &:= \langle \mathsf{e}(n_b)_{(n_a,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{(b,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{(a,n_b)}\rangle \parallel \langle \mathsf{e}(n_b)_{k_{ab}}\rangle. \end{aligned}$$

4.5 Encoding for SMT-Solver

We will answer the question whether $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ holds, given two processes P_1 and P_2 in standard form $P_i = \nu\vec{x}_i.(\langle\Gamma_i\rangle \parallel Q_i)$ for $i \in \{1,2\}$. First, we will investigate the complexity of this question. Second, we will present how to encode this problem as an SMT instance.

4.5.1 Complexity

To start with, we consider the *pure* fragment of our π-calculus, i.e. we restrict ourselves to names and process calls. In [KM08], it was shown that structural congruence is graph isomorphism-complete for the pure π-calculus. We do not check structural congruence but (knowledge) embedding for $P_1 \sqsubseteq_{\mathsf{kn}} P_2$. Intuitively, this result suggests that checking embedding is subgraph isomorphism - complete.

Recall the definition of graphs. A graph is a two-tuple (V, E) where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A graph is called *undirected* if $\forall u, v\colon (u,v) \in E \implies (v,u) \in E$.

Definition 47 (Subgraph Isomorphism from [Weg05] [1] **).** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs. G_1 is *isomorphic* to a *subgraph* of G_2 if there is a subgraph of G_2, denoted by $G_2' = (V_2', E_2')$ with $V_2' \subseteq V_2$ and $E_2' \subseteq E_2$, such that there is a bijective function $f\colon V_1 \to V_2'$ with $(u,v) \in E_1 \implies (f(u), f(v)) \in E_2$.

[1]Note that the definition is not given precisely this way but we combined the definitions of subgraph isomorphism and isomorphism.

Lemma 32. Checking $\sqsubseteq_{\mathsf{kn}}$ is subgraph isomorphism-hard.

Proof. We show that subgraph isomorphism is Karp-reducible to checking $\sqsubseteq_{\mathsf{kn}}$. This proof is inspired by the one in [KM08]. Let $G_i = (V_i, E_i)$ be two graphs for $i \in \{1, 2\}$. We give a method $\mathcal{R}(\text{-})$ that constructs a pure π-calculus-term, given a graph. To this end, we introduce a name restriction v_j for every node and a process call definition $\mathsf{Q}_1[v_j]$. For every edge (v_j, v_k), we introduce $\mathsf{Q}_2[v_j, v_k]$. Note that for every two vertices that are connected, we will introduce two process calls $\mathsf{Q}_2[\text{-}]$ due to the way we defined that a graph is undirected. Let $V_1 = \{v_{1,1}, \cdots, v_{1,n}\}$ and $V_2 = \{v_{2,1}, \cdots, v_{2,m}\}$. Formally, $\mathcal{R}((V, E))$ with $V = \{v_1, \cdots, v_n\}$ is defined as follows:

$$\mathcal{R}((V, E)) = \nu v_1, \cdots v_n.(\textstyle\prod_{v \in V} \mathsf{Q}_1[v] \parallel \prod_{(v_j, v_k) \in E} \mathsf{Q}_2[v_j, v_k])$$

It remains to show that G_1 is isomorphic to a subgraph of G_2 if and only if $\mathcal{R}(G_1) \sqsubseteq_{\mathsf{kn}} \mathcal{R}(G_2)$:

$$\begin{gathered} P_1 := \nu v_{1,1}, \cdots, v_{1,n}.(\textstyle\prod_{v \in V_1} \mathsf{Q}_1[v] \parallel \prod_{(v_j, v_k) \in E_1} \mathsf{Q}_2[v_j, v_k]) \\ \sqsubseteq_{\mathsf{kn}} \\ P_2 := \nu v_{2,1}, \cdots, v_{2,m}.(\textstyle\prod_{v \in V_2} \mathsf{Q}_1[v] \parallel \prod_{(v_j, v_k) \in E_2} \mathsf{Q}_2[v_j, v_k]). \end{gathered}$$

($\Rightarrow$) If G_1 is isomorphic to a subgraph of G_2, there is a function $f\colon V_1 \to V_2$ such that $(u, v) \in E_1$ implies $(f(u), f(v)) \in E_2$ which we denote by $(*)$ by definition. Hence, we can rename the image of $f(V_1)$ in P_2. All process calls of shape $\mathsf{Q}_1[\text{-}]$ will be covered as V_1 is a set of names. By $(*)$, all process calls of shape $\mathsf{Q}_2[\text{-},\text{-}]$ will also be covered and the (knowledge) embedding holds: $P_1 \sqsubseteq_{\mathsf{kn}} P_2$.

($\Leftarrow$) Given that $P_1 \sqsubseteq_{\mathsf{kn}} P_2$, we know that there is a renaming for P_2 such that

$$P_2 \equiv \nu v_{1,1}, \cdots, v_{1,n}, w_1, \cdots w_{m-n}.(\textstyle\prod_{v \in V_1} \mathsf{Q}_1[v] \parallel \prod_{(v_j, v_k) \in E_1} \mathsf{Q}_2[v_j, v_k] \parallel Q)$$

for some remainder of process calls Q. This renaming induces an surjective function $f\colon V_2' \to V_1$ where $V_2' \subseteq V_2$. Let us consider the inverse function $f^{-1}\colon V_1 \to V_2$. First, it is injective. Second, for every edge $(u, v) \in E_1$, we know that $(f^{-1}(u), f^{-1}(v)) \in E_2$ which is the condition to be isomorphic to a subgraph of G_2. Hence, the claim follows.

□

Lemma 33. Provided that $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ can be decided in polynomial time for arbitrary Γ_1, Γ_2, checking $\sqsubseteq_{\mathsf{kn}}$ is **NP**-complete.

Proof. In Lemma 32, we have proven hardness for subgraph isomorphism. It is well-known that subgraph isomorphism is **NP**-complete in general [Weg05]. There are subclasses for which the problem is solvable in polynomial time but by the way we designed the translation, the pure π-calculus terms can model any graph. Hence, checking $\sqsubseteq_{\mathsf{kn}}$ is **NP**-complete. It remains to provide a non-deterministic polynomial algorithm to check $\nu\vec{x}_1.(\Gamma_1 \parallel Q_1) \sqsubseteq_{\mathsf{kn}} \nu\vec{x}_2.(\Gamma_2 \parallel Q_2)$. To this end, let us non-deterministically guess a renaming from $\vec{x}_1$ to $\vec{x}_2$. It is straightforward how to check $Q_1' \sqsubseteq_{\mathsf{kn}} Q_2$ for the renamed set of process calls Q_1'. By assumption, we can check whether $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ holds in polynomial time. □

Corollary 8. Checking $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ for the intruder model for symmetric encryption $\mathbb{I}_{\mathsf{sy}}$ is **NP**-complete.

Proof. The proof follows from Lemmas 29 and 33. □

Remark 7. Note that we encoded the subgraph isomorphism problem and not the *induced* subgraph isomorphism problem. For the latter, the set of edges between the vertices need to coincide and the problem can be solved in polynomial time for certain classes of graphs. However, the same as before applies: the transformation can handle very general graphs.

Which Solver Recall that our inclusion check is recursive and we will not only deal with one single check but several checks when checking inclusion. Additionally, knowledge is crucial for our setting and is important for an embedding to either hold or not. Therefore, we want the backend solver also to take care of knowledge. Hence, we refrain from employing a solver for induced subgraph isomorphisms but use an SMT solver. We will present how to encode constraints for renamings, process call and knowledge in an SMT instance.

4.5.2 Names

Conditions for a valid renaming:

- We define two sets for $\vec{x}_1$ and $\vec{x}_2$: X_1 and X_2.
- We are looking for an injective function $f : X_1 \to X_2$ matching the names.
- We require equality for global names.

4.5.3 Process Calls

Naive Approach For every definition Q, we use one function for the left and one for the right hand side which is constructed in the same way:

$$l_{\mathsf{Q}} : X^{\mathrm{ar}(\mathsf{Q})} \to \mathbb{N} \text{ and } r_{\mathsf{Q}} : X^{\mathrm{ar}(\mathsf{Q})} \to \mathbb{N}.$$

These functions indicate how often the process call with the given parameters are present on the left, respectively the right. For soundness, we could require that for every process definition Q:

$$\forall a_1, a_2, \cdots, a_{\mathrm{ar}(\mathsf{Q})} \in X_1 : l_{\mathsf{Q}}(a_1, \cdots, a_{\mathrm{ar}(\mathsf{Q})}) \leq r_{\mathsf{Q}}(fa_1, \cdots, fa_{\mathrm{ar}(\mathsf{Q})})$$

Even though this concept is sound, the number of constraints to add is unreasonably high.

Covering (Single) Process Calls We introduce one boolean for every pair of process calls with the same identifier which indicates whether this process call on the left is covered by the one on the right. In case it is used, this entails a specific part of the renaming. Technically, we introduce a boolean $B_{1,2}$ for every two process calls $(l, \vec{p}_1) \in Q_1$ and $(l, \vec{p}_2) \in Q_2$ with the same label l and list of parameters $\vec{p}_1$ and $\vec{p}_2$ and require $B_{1,2} \implies f\vec{p}_1 = \vec{p}_2$. Obviously, we add a constraint that every process call on the right is used at most once to cover one on the left. There are two ways to do this. First, we simply remember for every process call every boolean which indicates that it is used to cover and allow at most one to be true. Second, we could try to exploit how SMT solvers propagate information. To this end, the use of implications might be advantageous so that we know that choosing one boolean to be true entails others to be false. As before, we have to remember every boolean indicating that a process call is used. Let B be the set of these booleans and b be the new variable to be introduced. We add the following constraint for every $b' \in B$: $b \implies \neg b'$. Doing this in every step ensures that at most one boolean of the final set B can be set to true. To prove this, we consider some boolean b which was added to B_1 and is chosen to be true. We split the final set B in the following way: $B = B_1 \uplus \{b\} \uplus B_2$. Due to the constraints added for B_1, every $b_1 \in B_1$ is set to false. We know that for every $b_2 \in B_2$, $b_2 \implies \neg b$ was added to the constraints. In case b_2 was true, b would have to be false which contradicts the assumption.

4.5.4 Knowledge

Besides names and process calls, we also have to generate an encoding for knowledge which encodes the relation $\vDash$ we introduced in Section 4.2. We introduce a function

$$G : X_1 \cup X_2 \to \mathbb{B}$$

which indicates whether the ground message only containing the name is in the knowledge basis. The encoding for encrypted messages we do not know the key for generates multiple constraints. Intuitively, given an encrypted message on the left, we give all the options on the right with the same form, which is a disjunction of conjunctions. We also give the option to decompose and proceed recursively for the message and the key. In case of a pair, we immediately split as there is no message with this top level constructor.

Definition 48 (Format of a Message)**.** Let M be a message. We define the format of M to be the syntax tree of constructors $(\text{-},\text{-})$ and $\mathsf{e}(\text{-})_{\text{-}}$ needed to build a message from basic names.

Note that two messages of the same shape have the same format. Therefore, having the same format is a reflexive, transitive and symmetric order. Let us explain our approach for knowledge with an example prior to formalising it.

Example 25. Let us consider $\mathsf{e}(a, b)_{\mathsf{e}(c)_d}$.
There are two options how this message can be derivable on the right: First, there is a message with exactly the same format, e.g. $\mathsf{e}(x, y)_{\mathsf{e}(z)_w}$. Then, we require every single name to match:

$$a = x \ \wedge \ b = y \ \wedge \ c = z \ \wedge \ d = w.$$

Second, both components (a, b) and $\mathsf{e}(c)_d$ could be derivable independently.
For (a, b), we split again and observe that both are basic names so that we require that

$$G(fa) \wedge G(fb).$$

In case that these were no basic names, we would proceed recursively.
For $\mathsf{e}(c)_d$, we can again have a message with the same format $\mathsf{e}(j_1)_{k_1}$ and another one $\mathsf{e}(j_2)_{k_2}$ for which we introduce the following constraints:

$$(c = j_1 \wedge d = k_1) \vee (c = j_2 \wedge d = k_2)$$

as satisfying one constraint would be enough. As before, we can also split the message into its components and just require that

$$G(fc) \wedge G(fd).$$

We combine all these constraints and hence formalise the intuition:

$$\begin{array}{rl} & a = x \ \wedge \ b = y \ \wedge \ c = z \ \wedge \ d = w \\ \vee & ((G(fa) \wedge G(fb)) \wedge ((c = j_1 \wedge d = k_1) \vee (c = j_2 \wedge d = k_2) \vee G(fc) \wedge G(fd))). \end{array}$$

Guided by this example we construct a tree of constraints from which we derive one constraint by combining them in the right way which is illustrated in Fig. 4.1.

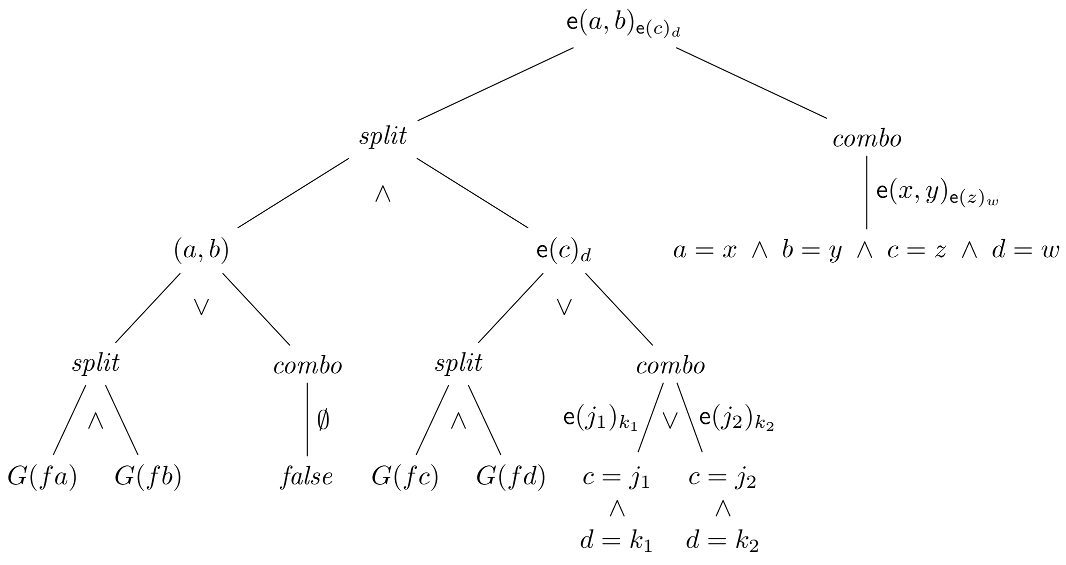


Figure 4.1: Tree of constraints for $\mathsf{e}(a,b)_{\mathsf{e}(c)_d}$

Let us just recall the setting briefly: We want to check whether $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ for two irreducible knowledge bases. As described in Algorithm 1, we consider every message M from Γ_1 and check whether we can compose it from Γ_2. For basic names, this is simple so we consider encrypted messages (and pairs as they might appear below encryptions). Therefore, the following algorithm can be generalised to intruder models satisfying the assumptions for Algorithm 1.

Algorithm 8 (Knowledge Constraints)**.** Let $M \in \Gamma_1$ be an encrypted message. We label the root of our tree with M and add two children with labels *combo* and *split*. These labels indicate what we do with M in the parent node and both subtrees are combined as a disjunction.

combo We take every message M' with the same format from Γ_2. There might be several M' which is why we add another level of children labelled by M'. As every single option is sufficient, we combine these to a disjunction.
For every M', we add a constraint that the names in the same positions of M and M' correspond to each other.

split Here, we consider both encrypted messages $\mathsf{e}(N_1)_{N_2}$ and pairs (N_1, N_2). First, we check the kind of message of N_i. In case of a basic message, we simply require $G(fN_i)$ to hold. In case of a pair or encryption, we add N_1 and N_2 as children and proceed recursively by building the tree of constraints for both of them.

In the actual encoding, we add a boolean variable for every node and add the corresponding constraints in case this option is chosen.

When investigating pairs in this algorithm, it is interesting to see that we consider them in *combo* for a concise presentation even though we know that there cannot be any message with the same format. This is due to the fact that $(\mathsf{-},\mathsf{-})$ cannot appear as a top level constructor by Lemma 26.

Preprocessing Checks There are some straightforward sufficient checks that can be employed to obtain a negative result faster, i.e. facts that have to hold for $P_1 \sqsubseteq_{\mathsf{kn}} P_2$:

- global names have to coincide.
- the number of name restrictions in P_1 needs to be smaller than the one in P_2.
- for every process call label in P_1, there have to be at least as many process calls with the same label.

Checking Congruences By checking $\sqsubseteq_{\mathsf{kn}}$ twice, we can also check $\equiv_{\mathsf{kn}}$ easily. Obviously, we merely have to do the checks if the number of fresh names and the number of process calls with the same label coincide. In order to check $\equiv$, we can have simpler constraints for knowledge that are analogous to the ones for process calls. Instead of checking for the same label, we check for the same format of both messages and require the renaming to match on the different positions of the message. As knowledge is persistent, there is no need to ensure that every message is only used once on the right. If there are no duplicates of messages in both knowledge bases, this will be the case anyway.

Potential Improvement The recursion of the inclusion check can lead to backtracking. Since we will deal with extensions of limits, there will be several "corresponding" solutions for mappings of names. Such mappings should not be considered again if the succeeding checks failed. There might be potential for improvement by ruling out corresponding name mappings via predicates, which is left for future work.

4.6 Tight Form

In contrast to the previous sections, we assume a general intruder model for this section. In Section 3.4, we have defined structural congruence for limits and explained that the presented rules enable us to swap names over ω's. The following two examples show two ways to amend limits and preserve their semantics.

Example 26. Consider the limit $L = \nu x.(\nu y.(\nu z.\mathsf{Q}[x,y] \parallel \mathsf{Q}[y,z])^{\omega})^{\omega}$ While it is obvious that $\mathsf{Q}[y,z]$ refers to the last name that was bound, i.e. z, there is no need for $\mathsf{Q}[x,y]$ to be in the same scope:

$$[\![L]\!] = \nu x.(\nu y.(\mathsf{Q}[x,y]^{\omega} \parallel (\nu z.\mathsf{Q}[y,z])^{\omega})^{\omega}$$

Let us recall the procedure for checking inclusion of two limits is recursive: to check whether $[\![L_1]\!] \subseteq [\![L_2]\!]$ holds, we unwrap one level of ω's in L_1 in every step of the recursion. It is straightforward to see that we hence want to minimise the ω-height overall.

Example 27. Consider the limit $L = \nu x, y.(\mathsf{Q}[x] \parallel \mathsf{Q}[y])^{\omega}$. It holds that

$$[\![L]\!] = [\![\nu x.(\mathsf{Q}[x])^{\omega} \parallel \nu y.(\mathsf{Q}[y])^{\omega}]\!].$$

So splitting and applying scope extrusion does not alter the semantics of the limit in this case. We can still amend the limit and observe that

$$[\![\nu x.(\mathsf{Q}[x])^{\omega} \parallel \nu y.(\mathsf{Q}[y])^{\omega}]\!] = [\![\nu x.(\mathsf{Q}[x])^{\omega}]\!].$$

As shown in the previous example, it can also be advantageous to split components to observe possible (inter-)dependencies. Thereby, the observation need not be that both limits are structurally congruent but rather whether one is included in another.

Overall, the intuition for the relation we want to establish follows these principles:

- every ω in L should contribute to the semantics of the limit, i.e. the limit has minimal ω-height for its semantics
$$\nexists L' \text{ with } \omega\text{-height}(L') < \omega\text{-height}(L) \colon [\![L']\!] = [\![L]\!]$$
- components should be "under" as few ω's as possible, i.e. the number of ω's on the path to components should be minimal
- the number of ω's in every level should be as high as possible while keeping the vector notation for name restrictions, i.e. let $L = \nu\vec{x}.(\langle\Gamma'\rangle \parallel Q' \parallel R')$ be a some limit in some level, then
$$\nexists L' = \nu\vec{x}.(\langle\Gamma'\rangle \parallel Q' \parallel R') \text{ with } |R'| > |R| \text{ and } [\![L]\!] = [\![L']\!]$$

Height vs. Width Inspired by the third principle, let us define ω-width which intuitively refers to the number of ω in parallel and can be thought of being orthogonal to height.

Definition 49 (ω-width)**.** Given a limit $\text{sf}(L) := \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$. We define width of a limit in standard form only on the top level: $\mathit{width}(L) := |R|$.

Remark 8 (Only Guidelines, No Normal Form)**.** Note that we cannot require all principles to hold pedantically. For instance, we can inflate the width in the third principle by adding the same sublimit arbitrarily. Overall, this means that we will not have a unique normal form for every limit with the same semantics but restrict it to the succeeding congruence relation.

Guided by the two examples and these principles, we define the following congruence relation for limits. The main goal is to have a coarser relation than structural congruence that preserves the semantics of limits.

Definition 50 (Limit Congruence)**.** Limit congruence $\equiv_{\mathbb{L}}$ is the smallest relation subsuming structural congruence $\equiv$ and satisfying the following rules:

$$(\nu\vec{x}.\nu\vec{y}.(P \parallel Q))^\omega \equiv_{\mathbb{L}} (\nu\vec{x}.P)^\omega \parallel (\nu\vec{y}.Q)^\omega \quad \text{if } \vec{x} \cap fn(Q) = \emptyset \wedge \vec{y} \cap fn(P) = \emptyset \tag{4.1}$$
$$(\nu\vec{x}.(P \parallel (\nu\vec{y}.Q))^\omega)^\omega \equiv_{\mathbb{L}} (\nu\vec{x}.P)^\omega \parallel (\nu\vec{y}.Q)^\omega \quad \text{if } \vec{x} \cap fn(Q) = \emptyset \tag{4.2}$$

Note that allowing $\vec{x}$ or $\vec{y}$ to be empty, (4.1) generalizes usual scope extrusion considering ω as context. (4.2) will not be applicable for limits in standard form as $\vec{x}$ is not used in its scope of restriction.

Lemma 34 (Correctness of $\equiv_{\mathbb{L}}$)**.** Let L_1, L_2 be two limits that are limit congruent to each other: $L_1 \equiv_{\mathbb{L}} L_2$. Then, $[\![L_1]\!] = [\![L_2]\!]$.

Proof. Let us sketch the proof. First, we know that structural congruence does not alter the semantics of a limit and hence it suffices to prove that a single application of each of both rules does not change the semantics of the limit. To this end, we consider expanded limits by Lemma 9 and $\sqsubseteq_{\mathsf{kn}}$-downward-closure. Let $n \in \mathbb{N}$ be some number. For Eq. (4.1), we can simply apply scope extrusion multiple times and obtain the same process on both sides. For Eq. (4.2), we observe that expanding with n on the left leads to process that is embedded by expanding the right hand side with $n * n$. □

We will present a function called *tigh*(-) computing a limit "satisfying" the presented guidelines. Equipped with this notation, we can introduce a preprocessing check for inclusion checks by exploiting the first principle.

Lemma 35. [Soundness of Preprocessing Check] For all limits $L_1, L_2 \in \mathbb{L}$ s.t. $[\![L_1]\!] \subseteq [\![L_2]\!]$, ω-height$(\mathit{tigh}(L_1)) \leq \omega$-height$(\mathit{tigh}(L_2))$.

We will prove this lemma at the end of this section. Let us formalise the principles we want to be guided by.

Restricted Form Let us recall an amended version of restricted forms for processes introduced in [Mey09]. However, we will see that these are too restrictive for our purposes.

Definition 51 (Fragments and Restricted Form from [Mey09])**.** Fragments, typically denoted by F or F_i, are defined inductively by

$$F ::= \mathsf{Q}[\vec{a}] \mid \nu a. \textstyle\prod_{i \in I} F_i$$

where $a \in \mathrm{fn}(F_i)$ for all $i \in I$. The set of all fragments is $\mathcal{P}_{\mathcal{F}}$. A process $P^{rf} = \prod_{i \in I} F_i$ is in restricted form. The set of all processes in restricted form is $\mathcal{P}_{rf}$ with $\mathcal{P}_{\mathcal{F}} \subseteq \mathcal{P}_{rf}$.

Note that we can omit the option of non-empty choices as choices are guarded in the calculus.

Example 28. Consider the following process with three name restrictions:

$$\nu x, y, z.(\mathsf{Q}[x, y] \parallel \mathsf{Q}[y, z] \parallel \mathsf{Q}[x, z])$$

We can transform this into

$$\nu x.(\nu y.\mathsf{Q}[x, y] \parallel (\nu z.\mathsf{Q}[y, z] \parallel \mathsf{Q}[x, z]))$$

but also every permutation of renamings of x, y and z. Note that this entails that restricted forms may not be unique.

Conglomerate of Name Restrictions As stated in the third principle, we want to keep the vector for name restrictions in general. Remember that we want to introduce a new (standard) form for limits for the application of the recursive inclusion check. In Section 4.5, we have explained how condition **(A)** can be encoded as an SMT instance. Hence, the solver will take care of the combinatorial problem of assigning name restrictions from left to right. The encoding only has a solution if a "partial" knowledge embedding exists, i.e. one that satisfies **(A)**. Therefore, it is reasonable to encode as many constraints as possible in one SMT instance in order not to backtrack more often than necessary. Overall, we want to keep the standard normal form for limits induced by structural congruence and only amend sublimits by limit congruence when considering ω.

To this end, we define a tight form for limits that is inspired by [Mey09] and the tied-to relation from [D'O15] that we lift to limits.

Definition 52 (tied-to (adapted from [D'O15])). We might want to define it on processes instead of indices for our purposes.
Let $\prod_{l \in L} A_l$ be all parallel components in standard form so that

$$\nu\vec{x}.(\langle\Gamma\rangle \parallel \prod_{j\in J_s} \mathsf{Q}_j[\vec{x}_j] \parallel \prod_{j \in J_\omega} R_j^\omega) = \nu\vec{x}.\prod_{l\in L} A_l$$

where $L = J_s \uplus J_\omega$ which means that we are considering single parallel components with A_l. A_l and A_m are linked if $fn(A_l) \cap fn(A_m) \cap \vec{x} \neq \emptyset$ which is denoted by $l \leftrightarrow_L m$. The *tied-to* relation is the transitive closure of $\leftrightarrow_L$: $l \frown_L m$, if $\exists n \in L$ s.t. $l \leftrightarrow_L l \wedge n \frown_L m$. We define an analogous connection for names in the sense that a name y is *tied to* A_l in L if $\exists m \in L. y \in fn(A_m) \wedge m \frown_L l$, written $y \lhd_L l$ and do not require $y \in \vec{x}$.

The goal of the tight form is to have a very similar shape as the standard form induced by structural congruence while exploiting limit congruence for some semantic observations.

Definition 53 (Tight Form). A limit L with standard form $\mathrm{sf}(L) := \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$ is called *tight* iff $\forall \mathsf{Q} \in Q, \forall y \in \vec{x}, y \lhd_L \mathsf{Q}$ and $\forall R_j \in R, \forall y \in \vec{x}, y \lhd_L R_j$ and R_j is in tight form.

4.6.1 Algorithm for Tight Form

Intuition: traverse the tree bottom-up and lift (bundles of) limits that do not use any name from the level above.

Definition 54 (Bundle). Every unsplittable limit L, i.e. there are no S_1, S_2 s.t. $L \equiv_{\mathbb{L}} (S_1 \parallel S_2)$, is called a *bundle.*

At first, we describe how to compute bundles of a limit, i.e. splitting it as far as possible so that it becomes a parallel composition of bundles.

Algorithm 9 (Computing Bundles). Given a limit $L = (\nu\vec{x}.(\prod_{j\in J} A_j))^\omega$ in standard form. We call names in $\vec{x}$ *recently bound* from the perspective of the components A_j.

Compute the subset of recently bound names used by every component A_j: $N_j \subseteq \vec{x}$.

Let us consider $\vec{x}$ as a set of nodes in an undirected graph. There is an edge between y and z iff there is a j s.t. $y \in N_j$ and $z \in N_j$. By definition, every name in $\vec{x}$ is used at least once in its scope of restriction which is why we cannot have isolated node. But there might be process calls that do not use any name in $\vec{x}$.

Then, every connected component of the graph and its associated components are one bundle: $P := (\nu\vec{x}_P.\prod_{j\in J_P} A_j)$. In case of isolated nodes, $\vec{x}_P$ is empty and $|J_P| = 1$.

For a limit L, we denote the set of bundles by $\mathfrak{B}_L$. Every single limit in $\mathfrak{B}_L$ is assumed to be in standard form. Recombining all bundles carefully leads to a limit which is limit congruent to L. To this end, let us define a function that reflects the intuition that no component can be decorated with two ω's. If no new names are bound, then the term is only decorated with an ω if it is no message and not already iterated.

$$w(P) := \begin{cases} P^\omega & \text{if } |\vec{x}_P| > 0 \quad \vee \quad |J_P| = 1 \wedge A_{J_P} \text{ is no message } \wedge A_{J_P} \neq C^\omega \text{ for some } C \\ P & \text{otherwise} \end{cases}$$

We combine the bundles using this function and obtain:

$$L \equiv_{\mathbb{L}} \prod_{P\in\mathfrak{B}_L} w(P)$$

where several applications of Rule (4.1) imply the congruence.

Lemma 36. For every bundle $B = \nu\vec{x}.\prod_{l\in L} A_l$, every name is tied to every component.

Proof. Towards a contradiction, assume that there is a name $x \in \vec{x}$ that is not tied to some component A_l. This means $\forall j \in L$ s.t. $x \in fn(A_j), A_l \not\leftarrow_B A_j$. This segregation entails at least two parts of $\vec{x}$ for which we could use Rule (4.1) to split the bundle. By definition, a bundle cannot be split which leads to a contradiction so the claim follows. Note that this contradiction corresponds to mixing up two (or more) connected components in the presented algorithm. □

Algorithm 10 (*tigh*(-))**.** This algorithm takes a limit L and transforms it into a parallel composition of limits in *tight form.* One can think of L as a syntax tree where vectors of names, messages and (iterated) process calls are nodes. Vectors of names should be the only kind of nodes to have children. We traverse the tree in depth-first search and apply the following steps after we visited every child of a node. From now on, we will refer to a node as the subtree rooting in it and the limit it denotes interchangeably.

1. For leafs, do not do anything.
2. For inner nodes, consider each one as a limit L and compute all bundles: $\mathfrak{B}_L$ and put $\prod_{P\in\mathfrak{B}_L} w(P)$ in the place of the original limit.
3. Now, we check which bundles have to be lifted. For every bundle, intersect all free names with all recently bound names (if existent).
   If non-empty, keep the bundle as limit on this level.
   If empty, lift the whole subtree one level up and put it in parallel with the other limits at this level. This transformation is sound due to (4.2) as no recently bound names as well as names from parallel sublimits are used.

By its recursive nature implemented as a depth-first search, one single application of *tigh*(-) suffices. Note that we chose to leave the bundles as computed and thereby amount to the third principle to increase width.

Corollary 9. Let L be a limit. Then, $[\![tigh(L)]\!] = [\![L]\!]$.

Proof. By construction of the algorithm, $tigh(L) \equiv_{\mathbb{L}} L$ and by Lemma 34, the claim follows. □

Example 29. $\nu x.(\nu y.(\nu z.\mathsf{Q}[x,z] \parallel \mathsf{Q}[y,z])^\omega)^\omega$ is a limit in tight form.

4.6.2 The Original Goal

Note that we wanted to decrease the computational workload for the recursive calls when checking inclusion for two limits. Moreover, we aimed at a quick preprocessing check whether two limits can even be included. Recall that we still need to prove that for all limits $L_1, L_2 \in \mathbb{L}$ s.t. $[\![L_1]\!] \subseteq [\![L_2]\!]$, ω-height$(tigh(L_1)) \leq \omega$-height$(tigh(L_2))$.

Definition 55. Let L and L' be two limits. We say that L' is *embodied* in L if there is a n-ary multi-hole context $\mathcal{C}[\bullet, \cdots, \bullet]$ such that

$$\mathcal{C}[L_1, \cdots, L_n] = L \qquad \text{and} \qquad \mathcal{C}[+, \cdots, +] = L'$$

where inputting $+$ in a context means omitting this hole completely and necessarily omitting parallel operators and respective name restrictions when becoming superfluous.

Computing a Witness of Height We can compute a witness of height, an embodied limit of every limit in tight form. The witness has the same height but only contains parts that contribute to the height either by being part of the longest path in the tree or contributing to the dependencies. There might be several occurrences of ω in the same level as the tied-to relation may need other sublimits to link.

Algorithm 11 (Witness of Height)**.** This algorithm takes a limit L in tight form with height h and returns an embodied limit L' of the same height. It is a recursive procedure.

For every level, only keep the bundle with maximal height and all (non-iterated) components that are tied to that bundle as well as only names that are tied to the bundle. Recurse and do the same for this bundle.

L' is not necessarily unique but gives a sequence of name restrictions $\vec{\vec{x}} = \vec{x}_1, \cdots, \vec{x}_h$ for the longest path in the syntax tree. For every level i, we know that some $x \in \vec{x}_i$ is used in the sublimit on the longest path. By its occurrence, we know that we can have an unbounded number of components using x and fresh names $\vec{x}_{i+1}$. Considering the next level, there is one name from $\vec{x}_{i+1}$ and so on. We define this component to be $C(x, x')$ for every $x' \in \vec{x}_{i+1}$.

Equipped with this algorithm, we conclude with the remaining proof of Lemma 35.

Lemma 35. [Soundness of Preprocessing Check] For all limits $L_1, L_2 \in \mathbb{L}$ s.t. $[\![L_1]\!] \subseteq [\![L_2]\!]$, ω-height$(\mathit{tigh}(L_1)) \leq \omega$-height$(\mathit{tigh}(L_2))$.

Proof of Lemma 35. Let $L_i^t := \mathit{tigh}(L_i)$ and $h_i := \omega$-height(L_i^t) for both limits. Given that $[\![L_1^t]\!] \subseteq [\![L_2^t]\!]$, we want to show that $h_1 \leq h_2$. We proceed by contraposition. We assume that $h_1 > h_2$ and show that the inclusion breaks. By the above procedure, we can compute the witness W_1 of height h_1 whose semantics are included in the the semantics of L_1. By its structure, W_1 provides us with vector of vector of names $\vec{\vec{x}} = \vec{x}_1, \cdots, \vec{x}_{h_1}$ for the longest path of the limit considered as syntax tree. On every level i, there is at least one $x \in \vec{x}_i$ that is used in the sublimit on the longest path due to the tight form. Thereby, we know that we can have an unbounded number of components using x_i and fresh names $\vec{x}_{i+1}$. Considering the next level, there is one name x_{i+1} from $\vec{x}_{i+1}$ and so on. We define this component to be $C(x_i, x_{i+1})$ for every i.
Starting from the other side, we use Lemma 14 and have to prove that $\exists n, \forall m, \lceil L_1^t \rceil^n \not\sqsubseteq_{\mathsf{kn}} \lceil L_2^t \rceil^m$. The idea for the contradiction is to pump the left side up with some number n so that it is impossible to find an m for the knowledge embedding. Let n be the sum of all parallel components on all levels of L_2^t. This ensures that instances produced from an enclosed limit of L_1^t cannot be fully covered by fixed parts of L_2^t. From before, we have a sequence of names $x_1, \cdots, x_{h_1}$ such that for every $i \in [1, h_1)$, we can have n times $C(x_i, x_{i+1})$ with fresh names x_{i+1}. Let m be any number with which we expand the right limit while we expand the left one by n. Now, we try to find an embedding. Because of the size of n, we always have to match a former sublimit to a former sublimit as the fixed part can never cover all the new fresh names. We do not necessarily have to choose the highest level possible for the matching but it definitely has to be a lower one than before. Hence, the highest would be the best case. Nevertheless, we will have to map at least two former sublimit-levels to the same one on the right as $h_1 > h_2$. Without loss of generality, let i and $i + 1$ be these levels. Then, n times $C(x_i, x_{i+1})$ and $n * n$ times $C(x_{i+1}, x_{i+2})$ ought to be covered on the same level for the appropriate fresh names. No expansion of L_2^t can introduce so many new names as every new sublimit can only introduce one layer of new names but we would require to get even new ones for the recently produced ones x_{i+1}. It is worth noting that only the tight standard form enables us to obtain this vector of names and the components $C(\text{-},\text{-})$ or rather enables us not to get them in L_2. Hence, there is no such m and the inclusion breaks which is a contradiction. □

Chapter 5

Evaluation

In this chapter, we present the evaluation of our approach of inductive invariants for cryptographic protocols. We built a proof-of-concept prototype tool that implements the approach for the intruder model for symmetric encryption $\mathbb{I}_{\mathsf{sy}}$. As most algorithmic aspects have been discussed in Chapter 4, we only briefly discuss the setup. We present a benchmark suite consisting of variants of well-known protocols and the corresponding empirical measurements obtained by using our prototype implementation.

5.1 Benchmarks

The prototype comprises means to parse limits for which we can check inclusion as well as inductivity. For the latter, we exploit the incorporation check explained in Section 4.1. It is also possible to generate invariants starting from a non-inductive limit using the coarse widening from Section 4.4.

The tool currently only supports symmetric encryption but there are plans to extend the tool to support asymmetric encryption, signatures and hashing. Based on the observations for the generic use of the algorithms in Chapter 4, this should be feasible. The sourcecode and the benchmark suite with its protocol models can be accessed at `https://bitbucket.org/bordaigorl/lemma9`. The name "Lemma 9" used to be an internal name for a cornerstone component of this verification approach. Eventually, it became the "Absorption Axiom". Let us recall the benchmarks from [DS19]. We applied the approach to variants of well-known protocols: Needham-Schröder(-Lowe) (NHS, NHSL), Yahalom (YAH), Andrew's RPC (ARPC), Otway-Rees (OR), Kehne-Schönwälder-Landendörfer (KSL, KSLR with reauthentification). The results are summarised in Table 5.1. For every variant of the protocols, we tried to infer an inductive invariant by coarse widening. This worked for all examples except Needham-Schroeder-Lowe. There, we needed to interactively combine two different invariants. The result was then inductive. For this example, the time reflects solely the sum of the two generations. The second time reflects how long it takes if an inductive invariant is given as correctness certificate and inductivity is re-checked. In general, it is fair to state that the results look promising so that an application to real-world protocols does not seem unrealistic.

Table 5.1: Experimental results. Columns: **Infer**ence of invariant fully automatic (F) or interactive (I); **C**heck of inductiveness; **S**ecrecy proved (✓), not holding (×), not modelled (○).

Name	Infer	C	S	Name	Infer	C	S	Name	Infer	C	S
Ex.2	2.6s F	1.0s	✓	ARPC	0.4s F	0.1s	✓	NHS	8.5s F	1.7s	○
OR	4.0s F	2.1s	○	YAH	8.6s F	2.8s	○	NHSL	2.4s I	16.1s	✓
ORl	37.5s F	3.9s	×	YAHsI	13.4s F	2.8s	✓	KSL	45.0s F	11.2s	✓
ORs	15.0s F	2.3s	✓	YAHsII	9.7s F	2.2s	✓	KSLR	175.1s F	36.2s	✓

5.2 Otway-Rees Protocol

In Section 4.4, we already discussed the Yahalom protocol which is why we only discuss one interesting example here: the Otway-Rees protocol. It exemplifies that the size function for variables in patterns is important for soundness. Recall the description of the protocol as given in [DOT17].

$$
\begin{array}{lll}
(1) & A \to B: & M, A, B, \mathsf{e}(N_A, M, A, B)_{K_{AS}} \\
(2) & B \to S: & M, A, B, \mathsf{e}(N_A, M, A, B)_{K_{AS}}, \mathsf{e}(N_A, M, A, B)_{K_{BS}} \\
(3) & S \to B: & M, \mathsf{e}(N_A, K_{AB})_{K_{AS}}, \mathsf{e}(N_B, K_{AB})_{K_{BS}} \\
(4) & B \to A: & M, \mathsf{e}(N_A, K_{AB})_{K_{AS}}
\end{array}
$$

Without Classification To start with, we consider the protocol without labels for secrets. The protocol can be encoded with the initial configuration:

$$\nu a, b, k_{as}, k_{bs}.(\mathsf{S}_1[a, b, k_{as}, k_{bs}] \parallel \mathsf{A}_1[a, b, k_{as}] \parallel \mathsf{B}_1[a, b, k_{bs}])$$

using the following definitions:

$$
\begin{array}{lcl}
\mathsf{S}_1[x, y, k_{xs}, k_{ys}] & := & \mathbf{in}(n_x, n_y, m : (m, x, y, \mathsf{e}(n_x, m, x, y)_{k_{xs}}, \mathsf{e}(n_y, m, x, y)_{k_{ys}})). \\
 & & \quad (\nu k_{xy}.(\langle \mathsf{e}(n_x, k_{xy})_{k_{xs}} \rangle \parallel \langle \mathsf{e}(n_y, k_{xy})_{k_{ys}} \rangle \parallel \mathsf{S}_1[x, y, k_{xs}, k_{ys}])) \\
\mathsf{A}_1[x, y, k_{xs}] & := & \tau.(\nu m.(\langle (m, x, y) \rangle \parallel \mathsf{A}_1[x, y, k_{xs}] \parallel \mathsf{A}'_1[x, y, m, k_{xs}])) \\
\mathsf{A}'_1[x, y, m, k_{xs}] & := & \tau.(\nu n_x.(\langle \mathsf{e}(n_x, m, x, y)_{k_{xs}} \rangle \parallel \mathsf{A}_2[m, n_x, k_{xs}])) \\
\mathsf{B}_1[x, y, k_{ys}] & := & \mathbf{in}(((m : \text{size}\,1) : (m, x, y)). \\
 & & \quad (\nu n_y.(\langle x \rangle \parallel \langle y \rangle \parallel \langle \mathsf{e}(n_y, m, x, y)_{k_{ys}} \rangle \parallel \mathsf{B}_1[x, y, k_{ys}] \parallel \mathsf{B}_2[n_y, k_{ys}])) \\
\mathsf{A}_2[m, n_x, k_{xs}] & := & \mathbf{in}(k_{xy} : (m, \mathsf{e}(n_x, k_{xy})_{k_{xs}})).\mathbf{0} \\
\mathsf{B}_2[n_y, k_{ys}] & := & \mathbf{in}(y, k_{xy}, m : (m, y, \mathsf{e}(n_y, k_{xy})_{k_{ys}})).\mathbf{0}
\end{array}
$$

Note the use of a helper definition $\mathsf{A}'_1[\text{-}]$ for $\mathsf{A}_1[\text{-}]$ which is needed for the generation of an inductive invariant. This restriction is used to guide the generation. It is straightforward to automatically linearise multiple name restrictions and restore the original process call definitions. Since this kind of coarse widening is an intermediate step towards more sophisticated techniques, it is more convenient to have the possibility to determine the linearisation manually for now.

The result is the following:

$$(\langle b \rangle \parallel \langle a \rangle \parallel \mathsf{S}_1[a, b, k_{as}, k_{bs}]^w \parallel \mathsf{B}_1[a, b, k_{bs}]^w \parallel \mathsf{A}_1[a, b, k_{as}]^w \parallel L_1^w)$$

$$
\begin{array}{lcl}
L_4 & = & \nu k_{xy}.(\langle \mathsf{e}(n_y, k_{xy})_{k_{bs}} \rangle \parallel \langle \mathsf{e}(n_x, k_{xy})_{k_{as}} \rangle) \\
L_3 & = & \nu n_x.(\langle \mathsf{e}(n_x, m, a, b)_{k_{as}} \rangle \parallel \mathsf{A}_2[m, n_x, k_{as}] \parallel L_4^w) \\
L_2 & = & \nu n_y.(\langle \mathsf{e}(n_y, b, a, b)_{k_{bs}} \rangle \parallel \langle \mathsf{e}(n_y, m, a, b)_{k_{bs}} \rangle \parallel \langle \mathsf{e}(n_y, a, a, b)_{k_{bs}} \rangle \parallel \mathsf{B}_2[n_y, k_{bs}] \parallel L_3^w) \\
L_1 & = & \nu m.(\langle m \rangle \parallel \mathsf{A}'_1[a, b, m, k_{as}] \parallel L_2^w)
\end{array}
$$

The inductivity of this invariant proves that the protocol is depth-bounded.

Classifying Secrets It is obvious that we want to classify the session key K_{AB} as secret. We do so by amending the definition of $\mathsf{B}_2[\text{-}]$ to have the continuation $\mathsf{Secret}[k_{xy}]$.

Now, it is interesting to investigate the result of the invariant generation:

$$(\langle b\rangle \parallel \langle a\rangle \parallel \mathsf{S}_1[a,b,k_{as},k_{bs}]^w \parallel \mathsf{B}_1[a,b,k_{bs}]^w \parallel \mathsf{A}_1[a,b,k_{as}]^w \parallel L_1^w)$$

$$\begin{array}{lcl}
S_6 & = & (\mathsf{Leak}[(a,a,b)] \parallel \mathsf{Leak}[(m,a,b)] \parallel \mathsf{Leak}[(b,a,b)]) \\
S_5 & = & (\mathsf{Secret}[(a,a,b)] \parallel \mathsf{Secret}[(m,a,b)] \parallel \mathsf{Secret}[(b,a,b)] \parallel S_6 \parallel \mathsf{Secret}[k_{xy}]^w) \\
L_4 & = & \nu k_{xy}.(\langle \mathsf{e}(n_y,k_{xy})_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_x,k_{xy})_{k_{as}}\rangle \parallel S_5) \\
L_3 & = & \nu n_x.(\langle \mathsf{e}(n_x,m,a,b)_{k_{as}}\rangle \parallel \mathsf{A}_2[m,n_x,k_{as}] \parallel L_4^w) \\
L_2 & = & \nu n_y.(\langle \mathsf{e}(n_y,b,a,b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_y,m,a,b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_y,a,a,b)_{k_{bs}}\rangle \parallel \mathsf{B}_2[n_y,k_{bs}] \parallel L_3^w) \\
L_1 & = & \nu m.(\langle m\rangle \parallel \mathsf{A}_1'[a,b,m,k_{as}] \parallel L_2^w)
\end{array}$$

Even though the session key K_{AB} was not leaked, there are several occurrences of $\mathsf{Leak}[\text{-}]$ in the invariant. Since we only declare a message to be a secret if B thinks it is the session key, B considers a triple to a the key in some cases. This is undesired and stems from a type of attack which is called type confusion. We can bypass this by restricting B only to accept messages of size 1 for keys by amending the following definition:

$$\mathsf{B}_2[n_y,k_{ys}] = \mathbf{in}((k_{xy} : \text{size}\,1) : \mathsf{e}(n_y,k_{xy})_{k_{ys}}).(\mathsf{Secret}[k_{xy}])$$

With this restriction, the following invariant is generated. It proves that the protocol does not leak a session key.

$$(\langle b\rangle \parallel \langle a\rangle \parallel \mathsf{S}_1[a,b,k_{as},k_{bs}]^w \parallel \mathsf{B}_1[a,b,k_{bs}]^w \parallel \mathsf{A}_1[a,b,k_{as}]^w \parallel L_1^w)$$

$$\begin{array}{lcl}
L_4 & = & \nu k_{xy}.(\langle \mathsf{e}(n_y,k_{xy})_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_x,k_{xy})_{k_{as}}\rangle \parallel \mathsf{Secret}[k_{xy}]^w) \\
L_3 & = & \nu n_x.(\langle \mathsf{e}(n_x,m,a,b)_{k_{as}}\rangle \parallel \mathsf{A}_2[m,n_x,k_{as}] \parallel L_4^w) \\
L_2 & = & \nu n_y.(\langle \mathsf{e}(n_y,b,a,b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_y,m,a,b)_{k_{bs}}\rangle \parallel \langle \mathsf{e}(n_y,a,a,b)_{k_{bs}}\rangle \parallel \mathsf{B}_2[n_y,k_{bs}] \parallel L_3^w) \\
L_1 & = & \nu m.(\langle m\rangle \parallel \mathsf{A}_1'[a,b,m,k_{as}] \parallel L_2^w)
\end{array}$$

Chapter 6

Related Work

In this chapter, we will discuss other automatic verification techniques for cryptographic protocols that also employ the *symbolic model*. We start by highlighting the beginnings of invariants as verification technique for cryptographic protocols as well as mention the use of type system in this context. Moreover, we will explain how other approaches (tools) are dealing with undecidability of the general problem.

6.1 Invariants

The use of invariants for verification of cryptographic protocols was pioneered by [Mcl95, Pau98].

In [Mcl95], protocols are modelled as standard linear-time logic properties. The properties to prove are also given as linear-time logic formulas. It is a valid mean to strengthen these formulas in a way that they become inductive. One can then manually prove that they hold and hence the security property is guaranteed.

In the approach proposed in [Pau98], interactions in a protocol are modelled as events in Isabelle/HOL. A run of a protocol is a trace of which infinitely many can exist. The methodology to prove security properties is based on induction on traces. This works by proving invariant properties of traces of the protocol manually but checked by the theorem prover HOL.

In general, invariants often form the bases for all kind of properties to prove. First, a number of "well-formedness" invariants are established. Then, one refines the set of feasible traces by establishing more and more precise invariants. For instance, a message k is never sent as plaintext can be such an invariant. More complex properties are then proven by appealing to these invariants. Our work can help to automate the proofs of (secrecy) invariants.

Let us now turn to the general question of how to deal with undecidability. As explained in Section 2.3, verification problems for cryptographic protocols become easily undecidable because of the Turing-completeness of the model. There are different approaches how to deal with this:

- restricting the model in a way that it becomes decidable — at the expense of proving a weaker claim
- having a semi-decision procedure that may not terminate for some instances
- over-approximating solutions which may cause false positives.

For our use case, restricting the model often means bounding the number of sessions. Since our approach focuses on unbounded number of sessions, we only briefly hint at tools pursuing this way. We will focus on the two major tools, i.e. ProVerif and Tamarin, that can also handle an unbounded number of sessions. They employ semi-decision procedures and over-approximations but there are some classes of protocols for which termination can be guaranteed. We also sketch how we could incorporate our techniques into these tools to extend the class of protocols they can handle or to speed up computation in general. Even though our approach may use some over-approximations when generating the invariants, we build a sound and complete theory supporting an unbounded number of sessions.

6.2 Bounded Number of Sessions

Bounding the number of sessions renders a lot of verification problems decidable, e.g. one can check whether equivalence or trace properties hold for protocols that apply a range of cryptographic primitives. Intuitively, equivalence properties determine whether two separate executions of a protocol can be distinguished by the environment. For instance, it is crucial for e-voting protocols that no one can distinguish whether two people have voted for the same party. Trace properties do not compare two executions but only refer to one single trace.

- The AVISPA tool [ABB$^+$05] has been designed to scale up to large scale Internet security protocols. Overall, the tool comprises several components and backend solvers. It employs model checking techniques for protocol falsification, i.e. finding an attack on the protocol. For the same purpose, they also apply constraint-solving to search for attacks. One can also encode a bounded unrolling of a protocol and the violation of a security property as SAT instance. The latter is given to a SAT solver and a model is translated back into a concrete attack if one is found. For secrecy properties, the intruder knowledge can be under- and over-approximated using a tree automata based approach that exploits the connection to regular tree languages and rewriting.

- The DEEPSEC tool [CKR18] was the first tool to decide trace equivalence and labelled bisimilarity when intruder models are modelled as subterm convergent destructor rewrite systems (see Section 7.2). They achieve this by encodings into second-order logic constraint systems and partitioning the solutions for these systems.

- The SPEC tool [TNH16] automatically checks equivalence properties for security protocols that are modelled in a variant of the π-calculus, which incorporates cryptographic primitives. Their equivalence checks basically amount to checking open bisimulation for two processes, i.e. checking whether the executions of two processes cannot be distinguished in any context.

- The AKISS tool [CCCK16] is a verification tool for trace equivalence of the class of so-called determinate protocols. As trace equivalence and observational equivalence coincide for this class, it can also verify observational equivalence properties. Traces of bounded number of sessions are modelled as first-order Horn clauses and equivalence properties are then checked by resolution procedures.

This is a non-exhaustive list of tools for a bounded number of sessions. Despite of the state space explosion problem that is inherent to model checking techniques, it is fair to state that the application of model checking has been quite successful in the domain of automated verification of cryptographic protocols. They also contribute efficient algorithms to handle different cryptographic primitives which might be useful for the extension of our tool.

6.3 Unbounded Number of Sessions

In addition to allowing non-termination and over-approximations, there are two major approaches to handle an unbounded number of sessions from the user perspective:

- incomplete or non-terminating procedures,
  e.g. ProVerif [Bla16], MAUDE-NPA [EMM06]
- relying on user interaction,
  e.g. Tamarin [MSCB13] may need helper lemmas,
  Cryptyc [GJ02] needs type information

These two classes are not distinct but they can overlap. Our goal is not to give a full survey of cryptographic protocol verification but to present the ideas of two examples, i.e. ProVerif [Bla16] and Tamarin [MSCB13], and to show how they differ from our results and how our results could be integrated in their approaches. We will also briefly discuss type systems in the context of verification of cryptographic protocols.

6.3.1 ProVerif

Similar to the class of depth-bounded protocols, the research for ProVerif was initially motivated by proving secrecy [Bla01]. It evolved over the last decade to a mature tool supporting a wide range of cryptographic primitives which are modelled by equations or rewrite rules. The security properties that can be verified include secrecy, authentification and observational equivalence properties.

Approach Protocols are modelled in a variant of the π-calculus. They are automatically translated to Horn clauses using some approximations to handle the infinite state space. In particular, any clause can be used an arbitrary number of times so that the number of replications is ignored. The properties to be proven are translated to derivability queries for this set of Horn clauses. With a resolution method, one checks whether a query can be derived. If not, the property holds. If it is derivable, there might be an attack. One attempts to reconstruct it and if successful, one has a concrete attack and hence the property does not hold. In case an attack cannot be reconstructed, the result is indistinct and one does not know whether the property holds. The latter can be the consequence from abstracting too rigorously, which results in over-approximations.

Termination The overall procedure may not terminate due to the resolution step involved since the derivability problem is undecidable in general. In 2005, Blanchet and Podelski have isolated a fragment of protocols for which they proved termination: the class of *tagged* protocols with a restricted set of cryptographic primitives. Intuitively, a protocol is tagged if every different application of a cryptographic operator is tagged with a distinct constant (name). Therefore, tagging a protocol is trivial and honest runs can still happen in the same way while attacks might not be possible anymore in the presence of tags. Generally speaking, tagging is another option to prevent type confusion attacks which we have remedied with a sizing function.

It was shown in [DOT17] that the class of tagged and the class of depth-bounded protocols are incomparable. Intuitively, this is easy to see: Example 2 is a depth-bounded protocol without tags while it is straightforward to construct infinite encryption chains in the presence of tags. Hence, it would be interesting to understand whether the class of depth-bounded protocols also entails guaranteed termination in ProVerif.

6.3.2 Tamarin

The Tamarin tool [MSCB13] is one of the few tools that was designed to support Diffie-Hellman exponentiation for an unbounded number of sessions by default [SMCB12]. It also supports cryptographic primitives modelled by subterm convergent equational theories as well as associative and commutative operators.

Approach Protocols are modelled as multiset rewriting systems while the properties of interest had been trace properties modelled by temporal first-order logic in the beginning. Later on, the tool was extended to handle equivalence properties inspired by the ideas used in ProVerif. The multiset rewriting system is translated into a labelled transition system. Then, invalidating a trace property amounts to finding a witness for the negation of the temporal first-order logic formula obtained from the trace property to prove. Hence, a trace property can be validated if no witness is found. For this process, user interaction might be required in form of helper lemmas. These involve the security properties to prove but may often involve loop invariants for protocols with loops — as done in program verification. Their constraint-solving techniques then prove the latter and exploits them for the actual security properties. They employ backward-search techniques and in order to make them work, the backward search may not terminate for two reasons [SMCB12]. First, because of loops as infinite application of these kind of rules might be applicable, in particular if the given loop invariant is too weak. Second, if the protocol serves as a so-called generic message deduction oracle, the normal form conditions may not achieve to eliminate enough redundant steps.

Pruning State Space [MSCB13] states that formulas can be used to restrict executions in specific ways. This is where our theory of invariants may be useful to prune the state space. Invariants over-approximate the reachable state space and hence restricting executions/checks only to these states is still sound. Even though there is a translator from an extension of the applied π-calculus to the input set for Tamarin [KK14], this incorporation is far from trivial and is hence left for future work.

6.3.3 Type Systems

There has been work on using type systems for the verification of cryptographic protocols [DKSH11, CCD15, CGLM17]. It is not straightforward to characterise the protocols for which these type systems apply. A comparison with the class of depth-bounded protocols is beyond the scope of this thesis. The methodology is very different in the sense that types tend to match syntactic patterns of the use of cryptographic primitives that can be proven safe. There are also advanced techniques that require heavy global side-conditions. The latter are frequently discharged using some kind of model checking/constraint solving. We speculate that our model checking techniques could help solving these side constraints in the future.

Chapter 7

Conclusion

7.1 Summary

Models We presented a cryptographic variant of the π-calculus, with which security protocols can be modelled. The model is Turing-complete in general but we presented the decidable fragment of depth-bounded protocols. Intuitively, they represent all processes from which we can only reach processes using finitely many nested name restrictions. This does not translate to finitely many name restrictions due to possibly infinite branching behaviour when considering the syntax tree. Visually, we are restricting depth but not breadth. The intruder model was not given directly but as axiomatisation. We also presented a specific intruder model which was used to exemplify different concepts.

Theory We proved that the class of depth-bounded protocols is a completion-post-effective class of well-structured transition systems [BFM18]. For this purpose, we presented a class of expressions that represents all downward-closed (and directed) sets of configurations of protocols. Furthermore, we have proven a characterisation for inclusion of two limits that leads to a direct (recursive) algorithm to check inclusion of two limits. Lastly, we showed how to compute the symbolic version of post(-), i.e. all successors of a downward-closed set. Overall, we explained which kind of properties can be proven directly when using our invariants. These ranged from basic properties like secrecy to known-plaintext attacks. Moreover, the invariants could be used to prune the search space in other tools. Facts obtained from the invariants could be of help to prove more sophisticated properties.

Practice In order to make the approach scale for our prototype, we presented several techniques of which some only apply to the intruder model supporting symmetric encryption but we hinted at requirements for generalisations. We showed how to infer invariants automatically using a widening approach that was promising when applied to our benchmark suite. Some techniques have been developed to handle knowledge and the pattern matching mechanism and generate all possible matchings for $\widehat{\text{post}}$. Exploiting the fact that there is a connection between the sets to check inclusion for, i.e. $\widehat{\text{post}}(L) \subseteq L$ for some representation L, led to a simplified and sound but incomplete incorporation test that could be used as first indicator whether the inclusion holds. With a first proof-of-concept prototype, we have been able to verify several toy examples of protocols. Although there is still a lot of work left to close the gap to real-world protocols but, our results are promising to keep on pursuing this path.

7.2 Future Work

7.2.1 More Specific Intruder Models

We only presented a model for symmetric encryption. In order to bridge the gap between our toy examples and real-world protocols, there are plans to support more cryptographic primitives like asymmetric encryption, hashing and signatures. Given in a similar style, most results from Chapter 4 should still hold and hence the incorporation into the tool should be feasible.

7.2.2 Relaxing the Intruder Axioms

The intruder axioms fail to capture some cryptographic primitives. Since XOR is an important operation in cryptographic protocols for devices with little computational power, we consider ways to relax the axiomatisation to support XOR for instance. The following example shows that our axioms are currently too strong to support XOR.

Example 30 (XOR fails)**.** Let $\otimes$ be the XOR function with the common bit-wise definition that we extend to vectors of bits and hence messages. The following statement holds:

$$a \vdash a \otimes c \otimes c.$$

However, this does violate the (Locality) axiom.

7.2.3 The Encryption Oracle

Recall that the encryption oracle is a pathological pattern that prohibits a protocol to be in the fragment of depth-bounded protocols. To this end, we also have ideas to have *symbolic messages* in combination with a *lazy intruder*. Intutively, the content of a message does not matter until a principal tries to match on it. So for the reduction semantics, we could have symbolic messages that are annotated with possible content and will only be concretised when necessary. This could remedy the encryption oracle problem and hence lead to an extension of the supported fragment of protocols. This is also the core reason why protocols modelling Diffie-Hellman exponentation are not depth-bounded. Hence, these considerations could also remedy this obstacle and lead to supporting Diffie-Hellman exponentation.

7.2.4 Comparison with Different Formalisations

In Chapter 6, we alluded that the way messages can be rewritten is different in Tamarin and ProVerif for instance. There are two standard assumptions we intend to recapitulate. It is not straightforward to compare our axiomatisation and systems with these properties but it would be interesting to understand the different capabilities and hence restrictions on applicability of our theory to their setting.

Finite Variant Property ProVerif only supports cryptographic primitives that are modelled by rewrite rules $\rightarrow$ and an equational theory $\mathcal{E}$ that satisfies the finite variant property [Bla16]. Such a system $(\rightarrow, \mathcal{E})$ is said to have the finite variant property if for each term t there is a finite set $\{t_1, \cdots, t_n\}$ of $\rightarrow$-normalised instances such that every instance of t normalises to an instance of some t_i modulo $\mathcal{E}$ [EMS08, CLD05]. For instance, the finite variant property was proven to hold for Abelian Groups and a theory of modular exponentation while it does not hold for the theory ACUNh (Associativity, Commutativity, Unit, Nilpotence, homomorphism) [CLD05].

Subterm Convergent Rewrite Systems Tamarin supports subterm convergent rewrite systems as well as Diffie-Hellman exponentiation, bilinear pairings, and associative and commutative operators [SSCB14]. Even though it would be interesting to compare our generic intruder model to all the possibilities they support. To start with, we would like to focus on subterm convergent rewrite systems and find a relation with our axiomatisation. A rewrite system is said to be subterm convergent if it is terminating and confluent and every right-hand side is either a subterm of the left-hand side or a constant [SMCB12].

7.2.5 Else-Branches

Our current cryptographic calculus only supports protocols that are linear in the sense that a participant cannot decide on the content of a received message to pursue a different path. But intuitively, this is desirable if one considers the following scenario: assume that it is not only important for us as participant of a protocol that the intruder does not obtain the value of a message but neither do we want him to know whether we realised that he has compromised a session. So given that we realise that this message cannot stem from our intended partner of communication, we still want to send some random message (our partner would realise that it is random) but the intruder would not realise that we know he compromised the session. This behaviour can easily be implemented using else-branches but it is not supported currently. Besides introducing this choice, this extension will probably also lead to the need of having unions of ideals for invariants rather than their parallel composition. This in turn would entail the need for new techniques to map contexts of two different ideals so that we have something similar to the incorporation check.

7.2.6 Protocol Repair

Tools incorporating formal methods are usually used in early design phases of a protocol. Hence, another direction which we would like to pursue considers protocol repair. Given a procotol, one starts to verify it. But it seems to be incorrect as one finds an attack or some invariant that contains a leak for example. Then, we want to analyse the attack/invariant and synthesize a modification of the protocol eligible for a new invariant that does not contain a leak/attack but is still inductive. Applying this to a range of examples could help to spot common patterns that lead to a misuse of logic in protocol design and hence avoid flaws in an early design phase.

List of Figures

2.1 Deduction rules for the derivability relation of $\mathbb{I}_{\mathsf{sy}}$ 19

4.1 Tree of constraints for $\mathsf{e}(a, b)_{\mathsf{e}(c)_d}$ 74

List of Tables

5.1 Experimental results 82

Bibliography

[ABB$^+$05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115. ACM Press, 2001.

[BFM18] Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Information and Computation*, 258:28–49, 2018.

[Bla01] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[CCCK16] Rohit Chadha, Vincent Cheval, Stefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computional Logic*, 17(4):23:1–23:32, 2016.

[CCD13] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, 2013.

[CCD15] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *CSF*, pages 170–184. IEEE Computer Society, 2015.

[CCZ10] Hubert Comon-Lundh, Véronique Cortier, and Eugen Zalinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Transactions on Computional Logic*, 11(2):9:1–9:42, 2010.

[CGLM17] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *ACM Conference on Computer and Communications Security*, pages 409–423. ACM, 2017.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy*, pages 529–546. IEEE Computer Society, 2018.

[CLD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[DKSH11] Morten Dahl, Naoki Kobayashi, Yunde Sun, and Hans Hüttel. Type-based automated verification of authenticity in asymmetric cryptographic protocols. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2011.

[D'O15] Emanuele D'Osualdo. *Verication of Message Passing Concurrent Systems.* PhD thesis, University of Oxford, 2015.

[DOT17] Emanuele D'Osualdo, Luke Ong, and Alwen Tiu. Deciding secrecy of security protocols for an unbounded number of sessions: The case of depth-bounded processes. In *CSF*, pages 464–480. IEEE Computer Society, 2017.

[DS19] Emanuele D'Osualdo and Felix M. Stutz. Decidable inductive invariants for verification of cryptographic protocols with unbounded sessions. Technical report, 2019. `https://www.emanueledosualdo.com/doc/papers/inductive-invariants-security.pdf`.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[EMM06] Santiago Escobar, Catherine A. Meadows, and José Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.

[EMS08] Santiago Escobar, José Meseguer, and Ralf Sasse. Effectively checking the finite variant property. In *Rewriting Techniques and Applications*, pages 79–93. Springer, 2008.

[FG09] Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In *STACS*, volume 3 of *LIPIcs*, pages 433–444, 2009.

[Frä86] Roland Fräissé. *Theory of Relations.* Studies in Logic and the Foundations of Mathematics. North Holland, 1986.

[Frö15] Sibylle B. Fröschle. Leakiness is decidable for well-founded protocols. In *POST'15*, pages 176–195, 2015.

[GJ02] Andrew Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *CSFW*, volume 12, pages 77 – 91. IEEE, 2002.

[KK14] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. In *Symposium on Security and Privacy.* IEEE, 2014.

[KM08] Viktor Khomenko and Roland Meyer. Checking pi-calculus structural congruence is graph isomorphism complete. In *ACSD.* IEEE, 2008.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[Mcl95] James W. Grayand John Mclean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *CSF*, pages 108–116. IEEE, 1995.

[Mey09] Roland Meyer. *Structural stationarity in the π-calculus.* PhD thesis, Carl von Ossietzky University of Oldenburg, 2009.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW*, pages 174–187. IEEE, 2001.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *CSF*, volume 25, pages 78–94. IEEE, 2012.

[SSCB14] Benedikt Schmidt, Ralf Sasse, Cas Cremers, and David Basin. Automated verification of group key agreement protocols. In *Symposium on Security and Privacy*, pages 179–194. IEEE Computer Society, 2014.

[TGD10] Alwen Tiu, Rajeev Goré, and Jeremy E. Dawson. A proof theoretic analysis of intruder theories. *Logical Methods in Computer Science*, 6(3), 2010.

[TNH16] Alwen Tiu, Nam Nguyen, and Ross Horne. SPEC: an equivalence checker for security protocols. In *APLAS*, volume 10017 of *Lecture Notes in Computer Science*, pages 87–95, 2016.

[Weg05] Ingo Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms.* Springer, 2005.

[WZH10] Thomas Wies, Damien Zufferey, and Thomas A. Henzinger. Forward analysis of depth-bounded processes. In *FoSSaCS*, volume 6014 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2010.

[ZWH12] Damien Zufferey, Thomas Wies, and Thomas A. Henzinger. Ideal abstractions for well-structured transition systems. In *VMCAI*, volume 7148 of *Lecture Notes in Computer Science*, pages 445–460. Springer, 2012.

Appendix A

A.1 Proof that Forest Encoding Preserves $\sqsubseteq_{\mathsf{kn}}$

Lemma 5. *Assume $Q_1, Q_2 \in \mathbb{S}_s^Y$ with $\text{nest}_\mathsf{v}(Q_1) \leq k$ and $\text{nest}_\mathsf{v}(Q_2) \leq k$. Then $\mathcal{F}[\![Q_1]\!]_k \sqsubseteq_\mathbb{F} \mathcal{F}[\![Q_2]\!]_k$ implies $Q_1 \sqsubseteq_{\mathsf{kn}} Q_2$.*

Proof from [DS19]. First we strengthen the statement: we can prove that if $\mathcal{F}[\![Q_1]\!]_k \sqsubseteq_\mathbb{F} \mathcal{F}[\![Q_2]\!]_k$ then $\text{sf}(Q_1) = \nu\vec{y}.R$ and $\text{sf}(Q_2) = \nu\vec{y}.\nu\vec{z}.(R \parallel R')$, which clearly implies $Q_1 \sqsubseteq_{\mathsf{kn}} Q_2$.

We proceed by induction on k. The base case is a special case of the induction step, so let us consider the latter first. Assume $\text{nest}_\mathsf{v}(Q_1), \text{nest}_\mathsf{v}(Q_2) \leq k+1$ and $\varphi_1 = \mathcal{F}[\![Q_1]\!]_{k+1} \sqsubseteq_\mathbb{F} \mathcal{F}[\![Q_2]\!]_{k+1} = \varphi_2$, and let $B = \text{supp}(\varphi_1) \cap \mathcal{B}_Y(s)$ and $C = \text{supp}(\varphi_1) \setminus B$ (note that $C \subseteq \mathbb{F}_{s,k}^{Y \cup \{x_{k+1}\}}$). Then, by definition, there is an injective function $f \colon \text{supp}(\varphi_1) \to \text{supp}(\varphi_2)$ such that for each $P \in B$, $P = f(P)$ and $\varphi_1(P) \leq \varphi_2(P)$; moreover, for each $\varphi \in C$, $\varphi \sqsubseteq_\mathbb{F} f(\varphi)$ and $\varphi_1(\varphi) \leq \varphi_2(\varphi)$. By definition of $\mathcal{F}[\![\text{-}]\!]$ we know that each $\varphi \in C$ is the forest encoding $\mathcal{F}[\![P_1]\!]_k$ for some subterm (α-equivalent to) $\nu x_{k+1}.P_\varphi$ of Q_1 with $\text{nest}_\mathsf{v}(P_\varphi) \leq \text{nest}_\mathsf{v}(Q_1) - 1 \leq k$. Similarly for Q_2 we have $f(\varphi) = \mathcal{F}[\![P_{f(\varphi)}]\!]_k$ for some subterm $\nu x_{k+1}.P_{f(\varphi)}$ with $\text{nest}_\mathsf{v}(P_{f(\varphi)}) \leq \text{nest}_\mathsf{v}(Q_2) - 1 \leq k$. We can therefore apply the induction hypothesis and get $\varphi \sqsubseteq_\mathbb{F} f(\varphi)$ implies that $\text{sf}(P_\varphi) = \nu\vec{y}_\varphi.R_\varphi$ and $\text{sf}(P_{f(\varphi)}) = \nu\vec{y}_\varphi.\nu\vec{z}_\varphi.(R_\varphi \parallel R'_\varphi)$. As a consequence of this and of $\varphi_1(\varphi) \leq \varphi_2(\varphi)$, we have

$$Q_1 \equiv_{\mathsf{kn}} \Big(\prod_{P \in B} P^{\varphi_1(P)} \parallel \prod_{\varphi \in C} (\nu x_{k+1}.\nu\vec{y}_\varphi.R_\varphi)^{\varphi_1(\varphi)}\Big)$$
$$Q_2 \equiv_{\mathsf{kn}} \Big(\prod_{P \in B} P^{\varphi_1(P)} \parallel \prod_{\varphi \in C} (\nu x_{k+1}.\nu\vec{y}_\varphi.\nu\vec{z}_\varphi.(R_\varphi \parallel R'_\varphi))^{\varphi_1(\varphi)} \parallel R'\Big)$$

which clearly entails the claim, by application of α-renaming and scope extrusion to get the two standard forms. In the base case, $C = \emptyset$ from which the claim follows straightforwardly. □

A.2 Properties of Knowledge

Lemma 15. *Let $\Gamma_1, \Gamma_2, \Gamma$ be sets of messages such that $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$. Then, $\Gamma, \Gamma_1 \leq_{\mathsf{kn}} \Gamma, \Gamma_2$.*

Proof. Let $\Gamma_1 = \{M_1, \ldots, M_n\}$. We have to show that for all N, if $\Gamma, \Gamma_1 \vdash N$ then $\Gamma_2 \vdash N$. We apply (Cut) n times obtaining

$$\dfrac{\Gamma,\Gamma_2 \vdash M_1 \qquad \dfrac{\Gamma,\Gamma_2,M_1 \vdash M_2 \qquad \dfrac{\dfrac{\dfrac{\Gamma,\Gamma_2,(\Gamma_1 \setminus M_n) \vdash M_n \qquad \Gamma,\Gamma_1,\Gamma_2 \vdash N}{\vdots}\;\textsc{Cut}\quad \iddots}{\Gamma,\Gamma_2,M_1,M_2 \vdash N}\;\textsc{Cut}}{\Gamma,\Gamma_2,M_1 \vdash N}\;\textsc{Cut}}{\Gamma,\Gamma_2 \vdash N}\;\textsc{Cut}$$

From $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ we have $\forall i \leq n\colon \Gamma_2 \vdash M_i$, which implies, by (Mon), all the left-most premises. By (Mon), from $\Gamma, \Gamma_1 \vdash N$ we know $\Gamma, \Gamma_1, \Gamma_2 \vdash N$, which completes the derivation showing $\Gamma, \Gamma_2 \vdash N$. ☐

Corollary 1. Let $\Delta_1, \Delta_2, \Gamma_1, \Gamma_2$ be sets of messages s.t. $\Delta_1 \leq_{\mathsf{kn}} \Delta_2$ and $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$. Then, $\Delta_1, \Gamma_1 \leq_{\mathsf{kn}} \Delta_2, \Gamma_2$.

Proof. Two applications of Lemma 15 suffice: $\Gamma_1, \Delta_1 \leq_{\mathsf{kn}} \Gamma_2, \Delta_1 \leq_{\mathsf{kn}} \Gamma_2, \Delta_2$. ☐

A.3 Properties of $\sqsubseteq_{\mathsf{kn}}$

Corollary 2. Let Γ be a set of messages, $P_1, P_2 \in \mathbb{P}$ two processes for which $\mathrm{sf}(P_i) = \nu\vec{x}_i.(\langle\Gamma_i\rangle \parallel Q_i)$ for $i \in \{1,2\}$ and $P_1 \sqsubseteq_{\mathsf{kn}} P_2$. Then, $\langle\Gamma\rangle \parallel P_1 \sqsubseteq_{\mathsf{kn}} \langle\Gamma\rangle \parallel P_2$.

Proof. Direct consequence of Lemma 15. ☐

Lemma 16. Let P_1, P_2 be two processes and $n \in \mathbb{N}$. If $P_1 \sqsubseteq_{\mathsf{kn}} P_2$, then $P_1^n \sqsubseteq_{\mathsf{kn}} P_2^n$.

Proof. For every single instance of both processes use the indicated matching for names and process calls. Considering knowledge, using Lemma 1 n-times is sufficient. ☐

Lemma 17. Let $L' \in \mathbb{L}$ a limit s.t. $L' = L^\omega$ for some $L \in \mathbb{L}$. Then, $(\lceil L' \rceil^n)^m \sqsubseteq_{\mathsf{kn}} \lceil L' \rceil^{m*n}$ holds for every $m, n \in \mathbb{N}$.

Proof.
$$(\lceil L' \rceil^n)^m = (\lceil L^\omega \rceil^n)^m = ((\lceil L \rceil^n)^n)^m = (\lceil L \rceil^n)^{m*n}$$
$$\lceil L' \rceil^{m*n} = \lceil L^\omega \rceil^{m*n} = (\lceil L \rceil^{m*n})^{m*n}$$
We know that $\lceil L \rceil^m \sqsubseteq_{\mathsf{kn}} \lceil L \rceil^{m*n}$ and the claim follows by Lemma 16. ☐

A.4 Solving the Example in the Introduction

We do not intend to leave the solution to the motivating example, in which the goal was to send a parcel via an untrusted post service, completely open. Even though we will not give a detailed answer, we want to argue about properties of an appropriate protocol to establish a secure communication with our friend in Chapter 1. A first possibility is the use of a key establishment protocol for which usually a trusted server is assumed with which each participant has a secure connection. We encountered such kinds of protocols, e.g. the Otway-Rees protocol. Modelling this kind of cryptographic primitives as keys and locks is quite straightforward if the server can generate fresh keys and locks and pass them to the participants. There are also methods to exchange keys without such a trusted server, e.g. the Diffie-Hellman key exchange protocol. Even though this cannot be easily adopted to the setting of physical locks and keys, it is frequently adopted to internet communication.